dilithium Press

How to use the TI-99/4A COMPUTER

BILL BREWER & JERRY WILLIS

How to Use the TI-99/4A

Bill Brewer and Jerry Willis

dilithium Press
Beaverton, Oregon

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data

Willis, Jerry.
How to use the TI-99/4A.

Includes index.
1. TI-99/4A (Computer) — Programming. 2. Basic (Computer program language) I. Brewer, Bill. II. Title.
QA78.8.T133W55 1984 001.64'2 83-23163
ISBN 0-88056-135-1 (pbk.)

Printed in the United States of America

dilithium Press
8285 S.W. Nimbus
Suite 151
Beaverton, Oregon 97005

Contents

Chapter 1

What Do We Have Here?

This book is written for the newcomer to the new and jam-packed world of information about personal computing. Introducing you to the fascinating world of the Texas Instruments 99/4A home computer with as little confusion as possible is our objective. Whether you have purchased a home computer and want easy words about how to use it, or whether you are shopping for (or merely curious about) a personal computer and think the TI-99/4A might be the one for you, we'd like to give you some information.

As owner, prospective owner, or user of a TI-99/4A, you are face-to-face with an exciting, nearly unique experience. The TI-99/4A is like no other personal computer in the world. For the price of a video game console, this computer gives you video game capability to enjoy during relaxing times and powerful computing capability to use for more serious tasks, such as learning, managing a business or a home, programming, writing, and paying bills.

At a current price of less than $100, the TI-99/4A is quite a bit of computer. Among under-$100 computers, it has several unique qualities.

Figure 1.1 The TI-99/4A and accessories.

Most important, it has very strong software support. TI currently markets more software in plug-in cartridges than any other computer manufacturer. Technologically, it is unique among low-cost personal computers: it has a powerful 16-bit microprocessor, and it can be expanded to include the features of much more expensive machines.

Face to face with the TI-99/4A, you'll discover that it has a full-size, typewriter-style keyboard. With 16 kilobytes (thousand characters) of read-write memory that can be increased to 48 kilobytes, it can be programmed in several popular languages, including BASIC, Pascal, and several special-purpose languages such as Logo and PILOT. Designed around its advanced, high-performance microprocessor chip, this computer has excellent color graphics and sound synthesis features, including optional voice-synthesis capability. It runs literally thousands of programs designed for it by Texas Instruments, independent software sources, and active groups of users. You can get this software in cartridge, on tape, or on floppy diskette.

We'll help you explore the TI-99/4A home computer through both your thoughts and your keyboard entry. We hope you will forgive us if we don't assume you are a computer scientist or an electronics wizard. We want to give you the complete story about the use—and the value—of this computer in down-to-earth terms, so you can make the best possible use of it.

More specifically, we have written the book with the following objectives in mind:

1. To introduce you to the TI-99/4A and its basic components, to what they do and how they work together. (Chapters 1, 9, and 10)
2. To provide you with an overview of what you can do with the computer. (Chapter 1)
3. To give you step-by-step instructions on how to set up, or install, your computer and to tell you how to get it up and running. (Chapters 2 and 3)
4. To give you information about how to load and save your own programs on cassettes and diskettes. (Chapters 4 and 5)
5. To provide a comprehensive introduction to BASIC, the standard computer language on this computer, and to describe the use of color graphics and sound synthesis. (Chapters 6, 7 and 8)
6. To give you information about the software available for the computer, to help you select the best programs for your needs, and to give you our best advice about where to buy software. (Chapter 9)
7. To provide information about accessories for your computer and how to select, buy, install, and use your accessories. (Chapter 10)

8. To tell you about other sources of information about your computer, such as magazines, books, user's groups, and computer-based information networks. (Chapter 11)

Following a brief history of TI (which is really a brief discussion of how the TI-99/4A developed), this chapter describes the potential uses of the computer and tells about how it works. But if your eagerness to start using your computer makes history and more general considerations something for later reading, we invite you to move ahead now to Chapter 2.

A LITTLE HISTORY

Until 1972, Texas Instruments was known only as a very successful manufacturer of electronic components such as transistors and integrated-circuit chips. It invented the integrated circuit in 1958, the single-chip microprocessor in 1970, and the single-chip microcomputer in 1971.

Behind the Scenes at Early TI

Out of the public eye, however, TI had joined with the I.D.E.A. Corporation to develop the world's first commercially produced transistor radio in 1954. In 1958, it developed the first fully transistorized television receiver, and in 1970, TI developed the first liquid crystal display (LCD) watch.

Only after development of the LCD watch did TI step into the public spotlight with the manufacturing and marketing of consumer products. In 1972, TI marketed its first calculator, a four-function hand-held machine called DataMath. In 1972, although Texas instruments was itself a large component-manufacturing company, its consumer products group was scarcely more than an infant whose fate was to grow up rapidly.

Calculators and the TI-99/4A Home Computer

Hand-held calculators manufactured and marketed by TI in the early seventies made use of electronic-component design and development already under way in older TI groups. Three or four integrated-circuit chips, or ICs, in the hand-held calculators replaced hundreds of transistor circuits in desk-top calculators. A standard integrated circuit could perform all the functions of 50 to 200 transistors, and it drastically reduced the labor, time, and cost required to manufacture a calculator. Today the technology of creating integrated circuits has reached the point where many hand-held calculators have only one large integrated circuit. ICs are routinely manufactured that replace tens of thousands of individual transistor circuits. And the same ICs that provided the foundation for the

calculator industry also shortly provided the foundation for the personal computer revolution.

TI moved into the field of personal computers with its announcement of the original TI-99/4 home computer in June 1979. The list price of the unit was $1150, including a color monitor. In June of the next year, TI began marketing the TI-99/4 computer without the monitor for $950, a $200 saving. Then, in May 1981, the improved TI-99/4A home computer was introduced at the significantly lower price of $525. TI found itself at the forefront of vigorous and highly competitive marketing that has made "computers for everybody" a reality. At a current price of $89 at some competitive department stores (complete with five hours worth of coursework at TI Learning Centers), the TI-99/4A is one of the best—if not actually *the best*—buy in personal computers.

Center Stage with the Present TI

TI's vigorously competitive pricing continues. During late summer of 1983, TI announced it was dropping the price of its expansion unit (including increased memory, a disk controller, and a disk drive) from $1200 to $550 (with a $99 software package thrown in for good measure).

Additional new developments at TI include its exclusive marketing of 108 educational software packages from Control Data Corporation's respected Plato system. TI has tapped Control Data's $900 million investment and 20 years of experience in computer-assisted instruction software, so the TI-99/4A becomes even more of a "live-in" teacher than it has been.

On the recreation and fun side, TI has begun developing and marketing a series of ten Milton Bradley games with voice recognition and voice synthesis. While TI has had games that talk to you for some time, it now has games you can hold discussions with.

WHAT CAN YOU DO WITH THE TI-99/4A HOME COMPUTER?

Many people are still skeptical about inexpensive computers. After so many years of hearing about business computers, we are accustomed to computers that cost millions of dollars and require teams of trained specialists to run them. To help you begin to see just how powerful the TI-99/4A home computer is, this section describes some of its major uses.

A Path to Computer Literacy

Scarcely 300 years ago, more than 97 percent of the adults in the world could not read. Of course, 300 years ago you didn't have to read to be a successful, contributing member of society. Reading and writing were, in fact, a specialized occupation in many societies. The scribes mentioned in

the Bible, for example, were the professionals of biblical society who could read and write.

Today we consider the ability to read and write essential. We try, through the public school system, to help every citizen become literate—that is, to be able to read and write. Considering our literacy, some futurists believe our society is becoming vastly more information-oriented. They think the use of computer power to convey and process information is rapidly becoming an essential skill for most of the population.

Just as reading and writing has become important over the past 300 years, computer use will change over the next three—or fewer—decades from a task for a small group of highly trained specialists into an essential task for all of us. We're likely to find we're not literate in a modern sense at all if we're not *computer literate*.

The TI-99/4A gives you an excellent path to computer literacy. The learning path may be as short—or as long and rewarding—as you choose. You can spend $200 to $300 on the basic unit to begin your learning. Add $100 for books and training material, and you have less than $400 invested. After a few months' work, you may decide to use the computer for more indirect and advanced learning through serious computer applications (several of which will be discussed later in this section). Then you can, for less than $600, expand your computer. It can take you further into the information age as it becomes a powerful computer capable of many important tasks in your home, business, or school.

Programming in BASIC and Other Languages

Many new computer owners use their computers for recreational applications, while others learn how to *program* the computer. It's small wonder that they do: when they program the TI-99/4A, they tell it to do *exactly* what they want it to do. When they rely on programs written by others, *others* tell it what to do.

Programming involves typing lines of instructions into the computer in a language it understands. BASIC is its primary, built-in programming language. BASIC stands for Beginners All-Purpose Symbolic Instruction code. Although there are many different computer languages in use today, BASIC is by far the most popular.

The BASIC language has about as many dialects as there are brands of personal computers, but the BASIC understood by the TI-99/4A is a fairly standard version. If you learn to program in the BASIC language on this computer, you will have little difficulty programming other personal computers that use BASIC.

Because features of the TI-99/4A extend past the limits of standard BASIC language, especially synthesized speech and moving-image graphics (sprites, TI calls them), TI has developed and marketed its Extended BASIC language in cartridge form. Besides supporting all the computer's advanced features, Extended BASIC has many of the features of business BASIC dialects normally available for much more expensive computers. This BASIC dialect allows sophisticated and powerful programming.

Chapters 6, 7, and 8 of this book discuss programming the computer in BASIC, and there are at least ten other books currently in print on programming the TI-99/4A in BASIC. In addition, TI packages two of its own books on its dialect of BASIC with the computer. The Extended BASIC cartridge also comes with a well-written book. TI further supports your learning of BASIC by marketing tutorial (computer-assisted) lessons on both BASIC and Extended BASIC on cassette tape and disk.

We will focus on BASIC in this book because it is the most popular language for small computers today. However, there are other languages available for your computer. A more advanced and more powerful language called Pascal is also available, and PILOT and Logo are fully supported by TI. PILOT is used to develop computer-assisted instruction programs, and Logo is a very popular language that lets children become familiar with computer programming while they draw pictures on a video screen.

Machine Language and Assembly Language Programming

You can also write programs for your computer in a computer language called TMS9900 machine language and a corresponding (but much easier to use) language called TMS9900 assembly language.

Machine and assembly languages are both versatile and powerful languages used by professional programmers to create video game cartridges, word processing programs, and other programs that make the most of the power and speed of the TMS9900 microprocessor chip. However, the languages are not easy to learn.

It's a good idea for you to start with BASIC and put off taking on machine language until you have some experience. Later, when you have the experience, you might use TI's helpful assembly language programming package and avoid the tedium and confusion of machine language programming. The editor/assembler package can help you create machine language programs along with the professionals.

Education and Computer-Assisted Instruction

Software marketed by TI shows extraordinary emphasis on education. Even without Plato software, TI educational programs outnumber any other type of program produced by the company.

Both the scope and quality of the educational software available for the TI-99/4A is amazing.

Traditional primary school subjects are well represented: reading, writing, and arithmetic programs provide schools and homes with high motivation and high tech instruction to help students perform better at early ages.

Video-game settings are used to motivate students in many of the programs. The enthusiasm that leads children to play countless hours of PAC-MAN (or TI's own Munch Man) to win, to score higher, and to seek greater challenges is adapted to support the more serious tasks of learning the three Rs. Even the use of joysticks puts children into the physical environment that combines fun with learning.

Many of the educational programs use the TI speech synthesizer to help children who can't read yet and to teach spelling, a task that is at best ineffective with displayed instructions alone. The learning of other skills is made much less dependent on reading skill: children with reading difficulty are not penalized by their failure to understand wirtten instructions when they learn math. The spoken word adds a powerful educational dimension to computer-assisted instruction. In many cases the voice from the computer makes the difference between learning and not learning.

TI educational software is excellent because most programs are developed in consort with publishing companies with vast experience in public school educational practices. Co-developers include Scott Foresman; Addison Wesley; Milliken; and several other respected, successful textbook publishers.

Educational software from the Plato system extends far past the primary level. Courses in geometry, biology, behavioral and earth sciences, prose literature and poetry, and economics support high-school, early college, and life-long learning.

Over and above traditional school subjects, TI software provides educational support in playing chess or bridge, in composing music, in maintaining weight control, in staying physically fit, and in many other practical and sophisticated areas of life.

Business and Professional Applications

Color graphics, sound and speech synthesis, and joystick connections make the TI-99/4A an ideal home or school computer. It is also a good (though not truly great) business computer. It has a full-size keyboard. It has a large and expandable memory. Disk drives and printers (needed for many business applications) available for it can store 10 million characters of information and print out the information with letter quality. Unfortu-

nately, the same thing that makes it an excellent educational and recreational computer severely limits it for business applications. In the technological tradeoff that gives you excellent color graphics for educational and recreational programs, the TI-99/4A's screen displays only 24 lines of 40 characters of text. Most "dedicated" business computers, in contrast, have a 24-line by 80-character video display, and they do not produce color graphics, excellent or otherwise.

Nevertheless, millions of people do use the TI-99/4A and other computers with similar screen format (the Commodore 64, the Apple II, and the ATARI 800, for example) for business applications.

Word processing is probably the most popular business use for the TI-99/4A. TI-Writer, a good word processing program, lets you compose, edit, correct, and update written materials with less effort and more speed than is possible on a typewriter. With TI-Writer you create a document on the screen of the video display, a place where changes are much more easily made than on paper. When you are ready, the computer can create a copy of the document on your printer. In addition, you can save the document you have created on a cassette or diskette and load it back into the computer later, when you want to revise it or to print out another copy.

TI-Writer requires an expanded system. You need a disk drive, extra memory, and a printer. See Chapter 9 for information on the extra equipment needed. Chapter 10 describes several of the word processing programs currently available.

The creation of *spreadsheets* is the next most popular business of this computer. Many business applications require the calculation of large tables or charts of related figures. Loan amortization tables, salary schedules, insurance fee tables, inventory order records, travel expense forms, job cost estimates, and organizational planning all require many tedious calculations. Electronic spreadsheet programs such as VisiCalc and SuperCalc have earned millions of dollars for their creators because so many business people need a convenient, quick way of doing such jobs. The TI spreadsheet program, created by one of the leading independent microcomputer software houses (Microsoft), is called Multiplan. Details about Multiplan are provided in Chapter 9.

Telecommunications

The fastest-growing area of interest for TI-99/4A owners is telecommunication. Telecommunication involves connecting your home computer to other computers by phone lines. Your phone then becomes a link to hundreds, perhaps thousands, of other computers. The computers you communicate with may be the TI-99/4As of friends or business

associates, or they may be large computers that have been programmed to provide all sorts of information services to personal computer owners.

On the personal side of telecommunicating, you can sit back in your easy chair, get a comfortable grip on the keyboard, and then do everything from pay bills to read weather reports for Colorado ski resorts. Telecommunicating lets you dial local area networks and community bulletin boards run by colleges, computer clubs, amateur radio clubs, and other special interest groups.

On the professional side of telecommunicating, you can tap the services provided by several national *information utilities* and view on your screen everything from stock market quotations to the current AP or UPI newswire. You can use your computer to purchase products, check on airline schedules, get reviews of current movies, and look for books or articles on topics of interest to you. Access to the *World Book Encyclopedia* is even available on one popular information utility called The Source. The following are the addresses of the most active sources for your obtaining telecommunicated information.

The Source, 1616 Anderson Road, McLean, Virginia 22102; (703) 821-6660

CompuServe, Inc., 5000 Arlington Centre Boulevard, Columbia, Ohio 43220 (614) 457-8600

Bibliographic Retrieval Services, 1200 Route 7, Latham, New York 12110 (518) 783-1161

DIALOG Information Services, 3460 Hillview Avenue, Palo Alto, California 94304 (415) 858-3810

If you need access to a particular type of information, the *Directory of Online Databases* (published by Caudra Associates) and the *Directory of Online Information Resources* (published by CSG Press) list hundreds of information utilities you can contact by computer.

The pioneer in information and communication services, The Source, is the host for TEXNET. With TEXNET, all of the information described above is available in a form specifically designed for the TI-99/4A. In addition, you can access libraries of programs with color graphics and sound and speech synthesis features, or you can transfer your own programs to others and transfer their programs to your computer. TEXNET also carries news about TI products and programs.

If you want to use your computer for telecomputing, you will need to have an RS-232 expansion accessory, a modem, the Terminal Emulator II telecommunications program marketed by TI, and an account number. The RS-232 accessory may be the same unit that connects your computer to your printer. A modem converts signals from the computer into audio

signals that can be transmitted over telephone lines. Modems cost from less than $100 to over $600, depending on the features they provide. The TI modem, manufactured by one of the leading modem manufacturers in the country (Novation), costs under $100. You obtain the account number from the information service when you sign up. With most information services, you are usually charged a one-time registration fee and thereafter are sent a monthly bill for the time you have used.

Recreation: Fun and Games

Probably 70 percent of all TI-99/4A owners use it mostly for the recreation and fun of video games. These games are available from TI and many other sources, including magazines, books, and independent software publishers. TI markets games in cartridges and on tapes and disks, and the TI BASIC reference manual you receive when you purchase the computer has some games you may type into the keyboard. Some magazines regularly publish at least one or two game programs for the TI-99/4A in each issue (see Chapter 10), and many small (and a few large) companies sell game programs. The best recreational programs take full advantage of the color graphics and the sound and speech synthesis features of the TI-99/4A. Most also let you use joysticks rather than the keyboard to move about on the screen.

NOW THAT WE KNOW WHAT WE HAVE

Now that we have some idea about what the TI-99/4A home computer is good for, let's move on to explore it from new directions. In Chapter 2, we'll look at the components of the TI-99/4A system from the perspective of setting them up. In Chapter 3, we'll look at some simple elements of programming. Then we'll...

But let's take it one step at a time, learning and clarifying as we go.

Chapter 2

Setup and Installation

Even the most basic TI-99/4A unit comes with accessories. There is a power transformer that converts power from an electric outlet into the low-voltage power the computer uses. There is also a modulator that connects the television to the computer; the modulator contains a switch box that lets you switch between regular TV reception and the computer display. This chapter will show you how to connect everything and adjust the television set for best picture quality. We'll also give you instructions for installing cassette recorders and other accessories.

CONNECTIONS TO THE WORLD

The TI-99/4A console (the main unit containing the keyboard and electronics) has connectors on the top panel, the back panel, and both side panels. A connector on the top panel accepts memory cartridges. Back panel connectors accept cables for power, video and audio output, and cassette recorder input and output. A connector on the left side accepts a cable from a wired remote controller (joystick), and a connector on the right side accepts cables or peripheral units that increase the computing power of the system.

Top Panel Connection: The Cartridge

There is only one connector on the top panel (keyboard) of the computer console, but it is an important one. This connector, behind the recessed door at the top right of the keyboard, accepts memory cartridges. In most cases, these cartridges contain programs stored in read only memory (ROM), and in a few cases, cartridges contain both programs in ROM and additional read-write memory to expand the processing power of the console.

You can purchase cartridges that contain educational programs, video games, home management and business software, and several other types of programs from TI and other sources. Such programs generally convert the computer ito a "dedicated" or special-purpose machine for the time they are inserted. Once a cartridge is inserted into the connector slot, the computer behaves upon power-up much as a video machine behaves when a coin is pushed into its coin slot.

Without cartridges, the TI-99/4A is a fairly powerful general-purpose personal computer with the BASIC programming language and the elec-

tronics for saving and loading programs built in. Let's check it out first as a general-purpose machine and later on we'll explore more dedicated uses of it through program cartridges.

Three Back Panel Connections

Across the back of your computer's console, you will find three connectors. The connector in the center is the power connector, which provides low-voltage power to the console from the console transformer, located in the power cord.

The connector to the right of the power connector, directly in line with the TI logo and the 1 key on the keyboard, is the video and audio output connector. A cable provided with your computer runs between the console and a standard television set, and includes a video modulator. You may buy an accessory cable to connect video and audio directly to a monitor for higher-quality pictures and sound than most television sets produce.

The third connector on the back panel of the console is for an optional cable leading to a cassette recorder for loading and saving programs and data.

The Power Connection

It is a good idea to hook up the power cord first. The power cord and the transformer in it deserve special attention.

Figure 2.1 Inserting a cartridge in the TI-99/4A.

A TI-99/4A transformer, according to TI, might be the cause of a house fire. At least two versions of the power cord and transformer assemblies have been packaged with this computer. Both versions have a six-foot long four-conductor cable leading from a black plastic transformer box to a four-conductor connector that provides power to the console. Only one of the versions is dangerous. The difference between the two versions is the way they conect to power outlets.

In an earlier, less desirable version TI calls its A version, there is no cable leading to a plug for a wall outlet. Instead, the plug is built into the transformer case. The wall outlet must provide physical support for the transformer, which is quite heavy and becomes quite warm, as well as provide electrical connection. TI literature advises you to attach the tranformer to the electrical outlet with the screw that holds the outlet cover plate on. If you have, and for some reason must use, this version of the power cable, avoid plugging the transformer directly into a wall outlet. Plug it into an extension cord and rest it on a nonflammable surface, to avoid attaching it so permanently to the wall and to minimize any chance of a fire.

Because of the fire hazard, TI is voluntarily providing a safety-enhancing accessory for the A version power cables. TI literature says you may call its toll-free numbers to receive the accessory—essentially a fused extension cord. However, one of your authors did just that within the first few days the deficiency was receiving wide media attention, and almost a year later he has still not received his accessory (but neither, we hasten to add, has his house burned down).

The newer, modified version of the power cable assembly does not contain a power plug built into the transformer case. An AC power line comes from the side of the transformer opposite the cable leading to the console. This line, in turn, has a plug that connects to an extension cord containing a fuse.

To attach the power cable to the console, plug the four-conductor connector into the corresponding four-pin connector in the rear panel of the console. Then plug the AC connector into the wall.

After connecting the power cable, check to see that the power switch at the lower left of the front vertical panel of the console is OFF. If you connect any accessories to the console while the power switch is ON, you risk damage to the console, the accessories, or both. When the power switch is positioned to the right, the power is ON, and the light-emitting diode (LED) to the left of the switch is lighted. When the switch is positioned to the left, the power is OFF, and the LED is not lighted.

The Television Connection

The TI-99/4A can be connected to two different types of video displays. As it comes from the box, it has all the equipment needed to display information on an ordinary television set. It will work on either a color or black and white set, although most people attach it to a color set to take advantage of its color graphics features.

Your computer can also be connected to a video monitor with an accessory cable available from TI (*gratis* if you buy a TI video monitor). A color monitor produces much sharper color images than an ordinary color television, but it generally costs much more.

The round connector for a television or monitor has five contacts in a semicircle at the bottom and a single alignment notch at the top. This kind of round connector, frequently used on American, Japanese, and European stereo systems, is called a 5-pin DIN connector. The standard six-foot cable with the 5-pin DIN connector on one end and the television video modulator on the other plugs into the DIN connector on the back of the TI-99/4A console. The wire in this cable carries video and sound signals, as well as power supply voltages, to the modulator. The modulator is essentially a "micropower television transmitting station." It combines the signals at the frequency of either channel 3 or 4, and it puts out the broadcast-like signal over the 300 ohm twin lead that connects to the antenna terminals of a television set.

A switch at the bottom right of the TV modulator selects the channel you want to use. If your city does not have a television station at channel 3 or 4, the switch can be set to either channel. If there is a local TV station on either channel 3 or 4, select the unused channel to minimize the interference with your computer by a broadcast station. If your city has stations on both channels 3 and 4, select the channel with the weaker signal.

Another switch on the front of the modulator selects between the computer signal and the TV signal from an antenna or cable. When you push the switch securely to the bottom, it effectively connects the computer to the twin lead. Pushed firmly to the top, the switch connects the antenna or cable to the twin lead.

To connect the TI-99/4A console to a television set, plug the DIN connector end of the cable provided with the computer into the DIN connector on the back panel. Disconnect the twin lead from your television set and connect it to the two terminals labeled ANTENNA on the left side of the modulator. Connect the spade lugs on the twin lead coming from the modulator to the screws from which you removed the antenna twin lead (the VHF screws). From now on, when you want to use your computer,

push the switch down; when you want to watch television, push the switch up.

If you plan to use a color monitor instead of a television set, you need the accessory cable that has a DIN plug on one end and the appropriate connectors on the other end for audio and video signals.

The TI-99/4A produces a good picture on most television sets. However, like all computers, it generates a lot of electrical *noise*, which can appear as "snow," crosshatches, and other patterns on a television screen. When you get your computer up and running, try moving the cords to see if one position reduces interference. If you plan to buy a small television to use with your computer, it's a good idea to take the computer to the store and try it out before making the purchase (or get the store to agree that you can return it if it doesn't work well). Better still, you might consider buying the TI color monitor since it eliminates the interference problems and provides superior video display.

The Tape Recorder Connection

If you plan to load programs stored on cassette tapes into the computer's memory or save programs for later use, you might want to connect one or two tape recorders now. (Or if you want to load and save programs with much greater speed and convenience, you might prefer to set up a disk-memory system, as we discuss below.)

TI markets a tape recorder for use with the TI-99/4A, although with a few precautions almost any relatively high-quality cassette recorder will load and save programs adequately. For best results, you should use a recorder with a tone control. Fewer loading errors result when the tone control of a recorder is adjusted for maximum treble, or high frequency, output.

Attaching cassette recorders to the home computer with the TI accessory cassette cable is straightforward. Plug the end of the cable with the nine-conductor connector into the corresponding connector on the back panel of the console. Be especially careful to use the nine-pin connector on the back of the console, *not the identical connector on the left side of the unit*. (The connector on the left side is for joysticks, which TI calls "wired remote controllers," as discussed below.)

At the other end of the cassette cable, plug the miniature phone plug at the end of the black wire into the REMOTE or MOTOR jack of your tape recorder. Now plug the phone plug at the end of the *red* wire into the MICROPHONE jack, and plug the connector at the end of the *white* wire into the EARPHONE or EXTERNAL SPEAKER jack.

If your cassette cable has a second pair of red and white wires, they may be connected to the MICROPHONE and EARPHONE jacks of a second recorder. A second recorder is useful for loading and saving programs while data produced by a program is being loaded and saved on the primary recorder.

With power provided to your recorder(s) from a wall outlet or batteries, your computer is ready to save and load programs and data.

The Left-Side Joystick Connection

If you plan to use your computer extensively for video games, you'll want to plug TI's wired remote controllers (joysticks) into the connector on the left side of the console.

The connector for joystick cables is a nine-pin connector identical to the connector on the rear panel for the cassette recorder cable. *Make sure that you have the joystick cable plugged into the nine-pin D connector on the left side panel of the console.* With this detail as with most of the computer world, a little double-checking saves a lot of grief and aggravation, damage and frustration, and *time* in the long run.

The Right-Side Expansion Connection

Behind a "secret" door on the right side panel of the console is a connector which, literally, provides your computer with access to and from the rest of the world. If you slip you fingernail beneath the extension from the sliding plastic door and lift up, you'll discover that the secret connection is a 44-pin connector at the edge of one of the console printed circuit boards. These connectors are much like arteries to the heart of the computer: they carry signals to and from the heart of the system, the microprocessor chip, as we'll discuss later.

The effect of these arteries is to extend the computer beyond the physical cabinet of the console. Additional memory, disk memory systems, communications devices, special programming units, speech synthesis devices, and other peripheral units attach to the expansion connector. But while it is simple enough *that* they attach to the connector, it is a longer and more complicated story to say *how* they connect. The story involves a "choo-choo train," a "heavy-metal" box, and, ultimately, an "intelligent bus." Without further waiting, let's ride right into the storyland of the TI-99/4A expansion.

A Choo-Choo Train Connection

If you try to talk about a "choo-choo train" peripheral with someone straightlaced at TI, you may not get very far. But if you talk to a more

verbal person in the company, your message will be quite clear. You'll be talking about one of several expansion accessories that TI designed literally to couple into each other and into the console through the expansion connector. Like cars on a train, they are self-standing units that extend as far out from the console as you need. Each of the accessories couples (connects) with the next in line by a 44-line printed circuit board edge connector, just as the processor does at the "head" of the train.

The photograph of a coupled-up choo-choo train (Figure 2.2) may save us several thousand words. The photograph shows, from left to right, a console, a memory-expansion unit, a speech synthesizer, a disk-drive controller, an RS-232 interface (for a printer, telephone-line modem, and

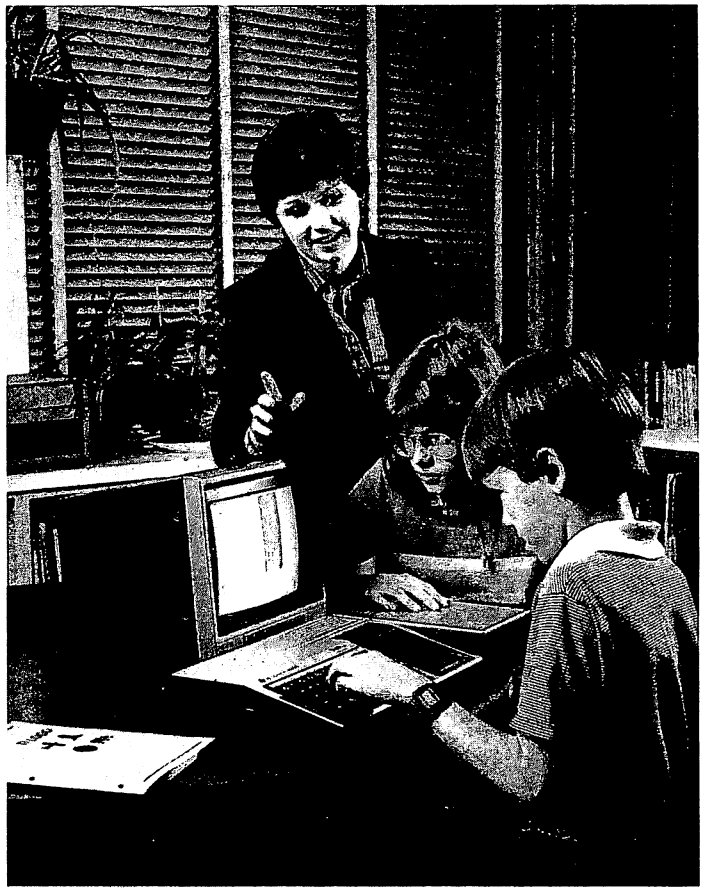

Figure 2.2 "Choo-choo train" style memory expansion.

the like), and a disk-drive unit. The maximum useful length of the train might be extended by two units: a second RS-232 unit and a *p-code* unit for running the computer with the Pascal programming language. (Beware that such a long train might not "run," because at such lengths of connection, signal interference and loss cause a great deal of difficulty. It may be necessary to remove the RS-232 units from the train, for example, while running Pascal.) Note that the memory-expansion unit is coupled immediately to the console. If the memory unit were farther from the console, noise and loss might make operation too erratic.

A Heavy-Metal Box Connection

Because of the obvious difficulty (and expense) with a long train of peripheral accessories, TI has offered an expansion box alternative. A single ruggedly constructed box is used to provide power, signal connection, and physical support for peripheral "cards." An expansion box is shown connected to the console in Figure 2.3. The accessory cards enclosed in it perform the same tasks as the units in the long train shown above. The inside of the expansion box contains a power supply to the left, connectors for up to eight peripheral cards in the center, and housing for a single disk drive to the right.

The expansion box is a definite improvement over choo-choo train packaging from several viewpoints. Overall manufacturing cost is less: a single supply powers all expansion accessories (except for disk drives other

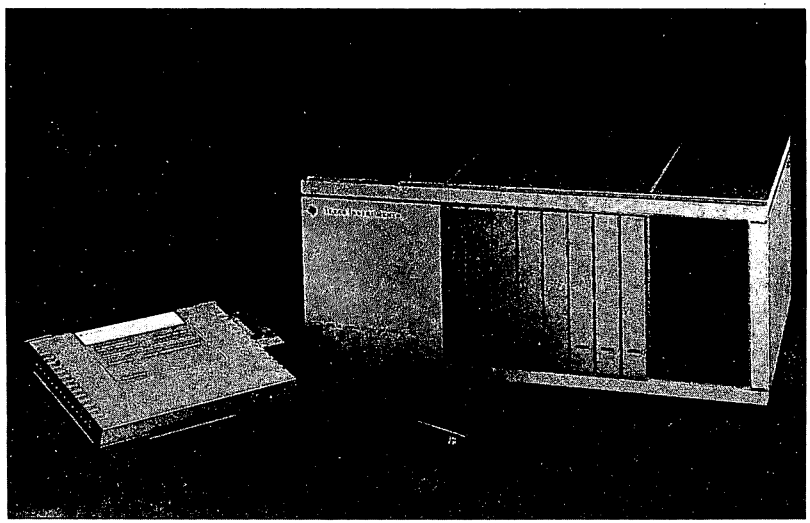

Figure 2.3 TI-99/4A Expansion box.

than the first one). The expansion box has a more manageable shape than the train-connected units, and it is physically smaller when many accessories are being used. Susceptibility to damage is less, because all peripheral cards and the main disk drive don't suffer the strain and fatigue of being moved about, unplugged and plugged. Overall, the electronic noise produced by peripherals in the box is much less than that produced by a functionally identical choo-choo train; the box is extremely well shielded, and television interference from the accessories within it is almost impossible.

The major drawback of most plug-in card systems (say the legendary S-100 system, for example) is avoided in the TI expansion box system. TI has simplified the handling and installation of cards to the point that it is only slightly more complex and worrisome than handling and installation of choo-choo train units. Each card is enclosed in a plastic case which even has recessed contacts: delicate chips on the card cannot be damaged by the static electricity that often accompanies handling them. The enclosure also serves as a guide for plugging in the card, making damage to the card-edge connector highly unlikely. Installing cards in the expansion box is easy.

It will take you only a few minutes and a very few cautions to connect an expansion-box system to your computer's console.

First, unpack the expansion box, the power cord, the signal cable, and the instruction booklet.

The expansion box packaging contains the expansion box itself, an AC power cord, a flat cable with large, enclosed printed-circuit connectors on each end, and an instruction booklet. But let us give you a couple of words of caution about unpacking. If you turn the packaging on its side to slide the expansion box out with the aid of gravity (which you are likely to do because of its weight and the tight fit of the box inside its styrofoam supports), *be careful*. If the heavier, power supply end of the expansion box is toward the top, and you turn it up too fast, the top of the box will keep on moving in an arc that's likely to include the bridge of your foot. One of the authors of this book (whose house didn't burn down and who doesn't have a permanent limp after all) provides this caution with good—though undisclosed—authority. Also, be careful to get the AC power cord from the packaging before discarding it. The cord is easily hidden beneath plastic sheet and styrofoam packing, and it might be lost.

Technically, the instruction booklet tells you that the expansion box end of the flat cable should be plugged into slot 1A of the expansion box. The back of the cable plug contains an attachment tab with a hole to match up with a sheet-metal screw in the back of the expansion box. This screw holds the plug securely in the expansion box.

If you are installing a disk drive and a disk controller, you need to observe another couple of cautions.

Remove the screw from the top of the disk housing, and turn the expansion box on end to remove an identically positioned screw from the bottom of the box. Pluck the "disk facade" from the front of the expansion box. Carefully save the screws for installation of the disk drive. The screw you removed from the bottom of the cabinet is quite a bit longer than the one you removed from the top. Remember that the longer screw must be used to attach the bottom of the disk drive, because this screw must extend through a false bottom in the drive housing to reach the drive.

For maximum space in the card area of the box (which is primarily convection cooled—the fan is only effective in cooling power supply components), the controller card should be installed in slot BA, the slot farthest to the right. When you have the card firmly installed, locate the ribbon cable from the controller package which has non-identical connectors on its ends. One end of the cable has a connector that slips over the card-edge connector of the disk drive; the other end contains a connector with two rows of 13 pinholes. The connector with the pinholes must be plugged—through a hole in the sheet metal sides of the disk-drive housing—into a matching connector at the back of the disk-controller card.

To install the cable connector to the controller card, bend the ribbon cable sharply at the connector so that it leads away from it in a straight line. While you are holding the connector with your fingertips, with the ribbon cable side of the connector to the front of the expansion box, reach into the disk housing past a small partition and plug the connector onto the pins on the back of the disk-controller card. You can see the plug from above the box, and a little eye/hand coordination and fishing experience come in handy. When the controller plug is firmly seated, slip the other connector onto the edge connector of the disk drive so that the ribbon cable extends across the back of the body of the drive. Insert the power connector on the four wires leading from the inside of the housing onto the connector at the bottom of the left side of the disk drive printed circuit board (the connector is keyed to fit only one way). Slip the drive into the housing with the red LED on the face of the drive to the upper left, letting the ribbon cable and the wires fold back into the box as they will.

When you have plugged in any other cards, the AC power cord, and the connector on the cable from the expansion box into the console, you are ready to try out a more powerful system.

You may be slightly disappointed to find that not all the peripheral units manufactured by TI are designed on cards to fit into the expansion

box. A few, notably the speech synthesizer and the more recently developed HEX-Bus controller, must still be connected in a relatively short train, with the expansion box connector being the caboose.

A HEX-Bus connection

A series of much smaller accessories have been designed recently by TI for the TI-99/4A and the new TI Compact Computer 40 (CC-40). These units are called HEX-Bus units because they connect to each other and a console with only six signal-carrying wires in cables that resemble modular telephone cables. No expansion box is required.

HEX-Bus peripherals announced by TI include an RS-232 communications unit, a modem, a compact four-color plotter/printer, and a wafertape mass storage unit. The units, each about a tenth the size of choo-choo train units, are appropriate for the TI Compact Computer console, especially since their flexible cabling allows them to be stacked, rather than connected side by side.

HEX-Bus units do not, however, plug directly into the expansion connector in the console. A separate, low-cost unit physically similar to the speech synthesizer must be plugged into the expansion connector, and the HEX-Bus cable leads out of this controller unit. The controller unit effectively translates the many signals on the 44 TI-99/4A expansion lines into the six-line signals used by the HEX-Bus units.

The most interesting of the announced HEX-Bus accessories is the wafertape peripheral. A *wafertape* unit, known in other quarters as a *stringy floppy*, is a program and data storage peripheral that combines much of the convenience of floppy-disk operation with the low cost of cassette-tape operation. Storage of programs and data is on *wafers*, endless-loop cartridges of 1/8-inch magnetic tape. The wafers are approximately 1 1/2 by 3 inches by 3/16th inch thick. Wafertape units are compact and significantly faster then cassette recorders (although they are significantly slower than floppy-disk units).

The four-color HEX-Bus printer/plotter is a fascinating device, although we believe that its major use will be as a compact-computer peripheral. It makes eye-opening small graphs on 2 1/2-inch wide (30 columns) paper, but its small print make it more novel than practical. Its smallness is a virtue in an executive's briefcase, it it is an unnecessary inconvenience on a home computer desk.

The photograph in Figure 2.4 shows a stack of HEX-Bus peripherals, including an RS-232 unit (bottom of the stack), a wafertape unit (middle of the stack), and a four-color printer/plotter (top).

After the HEX-Bus controller is plugged into the expansion connector of the TI-99/4A, one end of a small cable is simply snapped into the controller, and the other end is snapped into the connector of a peripheral. From that peripheral, an identical cable can lead to another, and from that to yet another...

The HEX-Bus accessories point to the future in TI-99/4A development. They are smaller, less expensive, and more convenient to work with than the expansion box or choo-choo train peripherals. But the HEX-Bus units are just coming into production, and until they are fully available, fully supported by software, competitively priced, and known to be highly reliable, the older peripherals are excellent choices for increasing the computing power of your computer.

Checking out the System

All the connections have now been made (unless you have something special or more powerful in mind for you computer and have other accessories to connect, in which case you might want to read about further connections in Chapter 4). With all systems intact, it's time to see if they are "go" for the many miles and years of computing that lie ahead.

Turn on the television and set the channel selector to either 3 or 4. Slide the channel selector switch on the bottom of the video modulator to the same channel. Also, *make sure that the switch on the front of the modulator is pushed all the way to the top*. The labeling on this switch is a bit

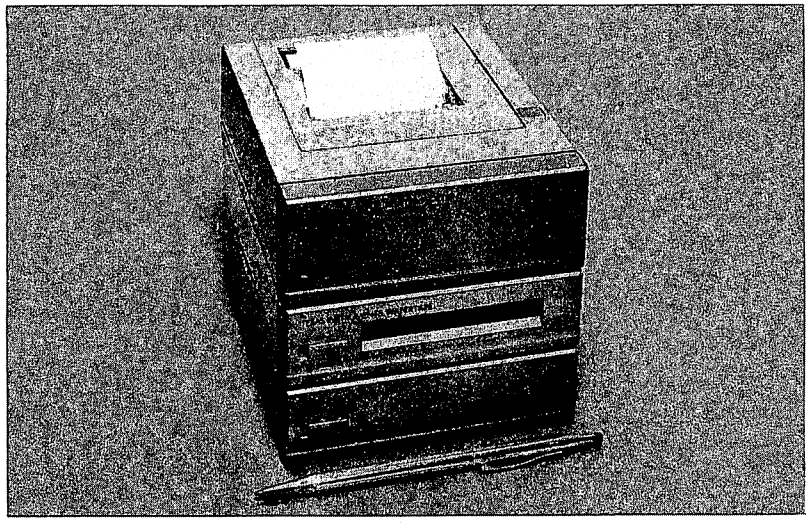

Figure 2.4 A stack of HEX-Bus Peripherals.

confusing; it says *Television* to the left of the top switch position. It does say MODULATOR in captials and inside a box of lines to the right, but the double message can be very confusing.

Turn on any peripheral accessories attached to your computer, and then turn on the console by sliding the ON/OFF switch, on the right side of the vertical front panel, to the right. *The console must be the last unit in an expanded system to be turned on.* The small red LED immediately to the left of the switch should now light up.

If you have made all the connections correctly, you should be looking at a video display screen similar to the one in the photograph in Figure 2.5. The photo shows the title screen that appears every time the TI-99/4A is turned on. If the title screen isn't being displayed, the source of difficulty will probably be easy to find. Go through the following list to find and correct any difficulty.

1. Check to see that power is supplied to both the console and the television set (or monitor). Power is supplied to the home computer console if the red LED beside the power switch is lighted, and power is supplied to the television set if the picture tube glows or the speaker emits sound. Because power is supplied to the speech synthesizer by the console, it is on when the console is on. Unfortunately, most of the peripheral units (including the expansion box) do not have power-on indicators, but you should be able to hear a "click" or "thud" from the television set when they are switched on. Remember that after you check to see if a peripheral unit is

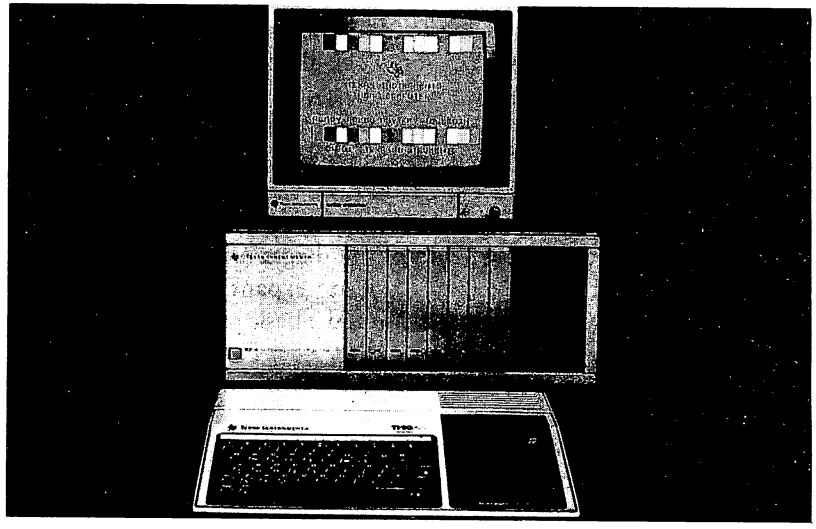

Figure 2.5 The TI-99/4A with Expansion Box, showing title screen.

on, you must reset the console by turning it off for a moment and then turning it back on. The console must be the last unit to be turned on in an expanded system.

2. When you have verified that power is supplied to all units in the system, test your wiring between the console, the modulator, and the television set. Your modulator should be set to the same channel your television is tuned to (either 3 or 4). Recheck all connections: the two connections on the twin lead from the modulator to the television and the five-pin DIN connection between the console and the cable to the modulator. Make sure that the switch on the front of the modulator is pushed firmly to the top (where, to the *right* of the switch lever the word MODULATOR appears). Finally, when all connections and modulator switching is verified, fine-tune the television set. A channel not usually used for reception may be out of tune to the extent that it will not receive a signal correctly broadcast on the channel. If during tuning you receive a picture that is wavy or fuzzy, use the fine tuning control further and the brightness/contrast controls to get as satisfactory a display of the title screen as possible. If all else fails, try another TV.

When you get a display that looks like the screen shown in Figure 2.5, you've reached the first major milestone, and all systems are "go". You'll check your system out further when you program your computer or run a program stored in cartridge, on tape, or on diskette (see Chapters 4 and 6)

THE ESSENTIAL TI-99/4A

Now that we have the units in the computer system connected and the title screen tells that all is well so far, we're almost ready to check it out through use. First, it might be well to take a quick look at your computer from an operational perspective. The TI-99/4A is a small but very powerful machine, and the power demands our understanding.

The TI-99/4A, like most small computers, can be viewed as a system made up of several basic components. Figure 2.6 shows a block diagram of the TI-99/4A. All of the elements in the diagram that lie outside the dotted line, except the keyboard, indicate expansion accessories (peripherals) you must obtain separately from the basic package.

The CPU

The CPU (central processing unit) is the heart of the system. Although CPU chips inside personal computers similar to the TI-99/4A (the ATARI 800 and VIC 20, for example) are about the size of a stick of chewing gum, the chip inside your computer is approximately twice that size. The chip

designation for the TI-99/4A is TMS9900. No other personal computer uses the TMS9900 chip; other computers use chips with designations such as Z80, 6502, and 6510. Some of the differences among these chips are of interest to computer designers and experienced programmers, but other differences are of interest to consumers and users. The primary matter of interest to us here is that the large TMS9900 CPU chip is, in general, faster and more powerful than the smaller chips in similar computers. The TI-99/4A CPU chip, for example, can execute machine language multiply-and-divide instructions with numbers as large as 65,535, while the other chips can only simulate multiplication and division through repeated addition and subtraction of numbers no larger than 255.

The CPU does most of the actual processing inside the computer. It sends signals and receives signals from other parts of the computer including *inputs* and *outputs* to and from the keyboard, the cassette-tape unit, and the television display. It also synchronizes the operation of each component in the system. This synchronization permits the system to perform hundreds of thousands of operations a second. These operations are fairly simple (for example, adding two numbers), but at such a speed the computer can combine them to do some very complex tasks in thousands of simple steps, still within an eyeblink.

Memory

When you enter a program from the keyboard or load one from a cassette, the program is placed in the computer's memory. Each letter or number you type on the keyboard is converted to a code and stored in the

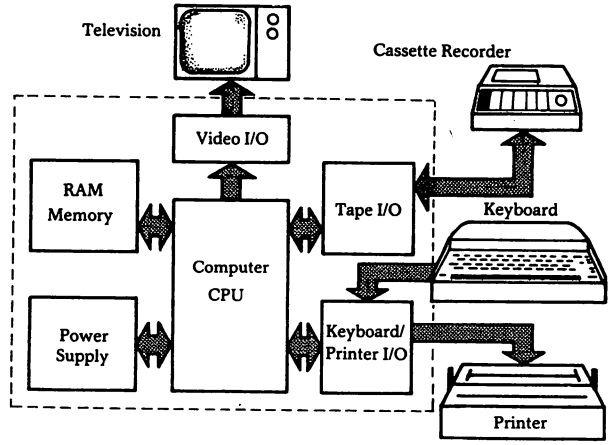

Figure 2.6 Block diagram of the TI-99/4A's components.

memory of the computer. The code consists of a series of ones and zeros (called *binary digits* or *bits*). A few of the codes are shown below:

Keyboard Character	Internal Code
A	1000001
B	1000010
1	0110001
2	0110010

All computers convert characters into codes of ones and zeros. (The ones and zeros are really electrical signals; negative voltage signifies a zero and a positive voltage signifies a one.) The code that is virtually universal today is ASCII, or American Standard Code for Information Interchange. Most small computers use this code for translating characters and symbols into signals that can be stored in memory.

In the ASCII code, as the example above shows, seven bits are used to define the code for each letter or character. Another bit, the eighth, may be added to the code for each character so the computer can check for errors. Whether this error-checking bit is used or ignored, the combined eight bits that form a character are called a *byte*. Bytes are the fundamental code units for information storage in the memory of the TI-99/4A and for most other small computers.

Your computer contains two types of memory: RAM and ROM. ROM stands for read only memory. The codes which ROM contains are built in at the factory: we can use them but we can't change them. the ROM inside your computer contains the thousands of machine language instructions the computer needs to run BASIC, and the ROM inside software cartridges contains instructions for the "computer's side" in playing a game.

If there were no preprogrammed ROM in the computer, you might have to give it instructions in the patterns of ones and zeros we've just discussed, a fate we would not wish on anyone. Fortunately, when you plug the computer in, it automatically begins to follow the instructions in its ROM. Instead of ones and zeros, we can use ordinary English words in the BASIC language, because TI programmers have already stored instructions in ROM to permit us to do just that.

Because ROM can only be coded at the factory, the computer must have another type of memory to accept our programs. Therefore, much of the memory in the computer is RAM, or random access memory. The standard TI-99/4A has a little over 16,000 bytes of RAM. Since each byte can store the code for one character, you would think the 16,000 bytes of RAM could store up to 16,000 characters of program and data. However, the TMS9900

processor must use some of its RAM for its own temporary storage or "housekeeping" work, so we don't have use of all the RAM. Our programs can use approximately 14,000 bytes.

RAM is also known as *volatile memory*. The contents of RAM go away when you turn off the computer or when you replace them with new contents. You can store data or instructions in RAM, tell the computer to use the information you've stored there, and then replace the material in RAM with something new. The biggest problem with RAM is the fact that whatever is in it disappears when the computer is turned off or whenever something new is put into it. If we need to save something we have in RAM so we can use it later, we must store it on a cassette or diskette before we turn the computer off or put something new in RAM.

You will see advertisements and magazine articles that refer to 16K, 32K, or even 128K of RAM for the TI-99/4A. Memory is often referred to in terms of K. Each K of memory is 1024 bytes. The 16K of RAM in the TI-99/4A console is, for example 1024 times 16, or 16,384 bytes. Just multiply the number of K by 1024 to determine the number of bytes of memory.

I/O Design

The abbreviation I/O stands for input/output. If a computer is to be of any use to you, it must be able to receive information and communicate its response back to you.

When the TI-99/4A provides information to you, it sends it out in one of several ways: on the television or monitor screen, on a printer, over communication (telephone) lines, through a television or monitor speaker by sound or speech synthesis, or onto a cassette or diskette for later use.

When you provide information to the home computer, you usually do so through the keyboard. The computer also receives information from communication lines and from a Milton Bradley speech-recognition unit, and it accepts previously saved information from cassette and diskette.

A cassette tape unit or disk drive connects to I/O circuitry that allows you to store programs on a standard audio cassette tape or floppy diskette. You can load the programs into the computer when you want to use them again.

Except for the keyboard, the display and sound-synthesis electronics, and the cassette-tape interface, all of the input/output circuitry for the TI-99/4A is outside the console. The I/O circuitry communicates with the CPU through the peripheral expansion connector on the right side of the console. But in the TI-99/4A, in contrast to most other personal computers, the instructions the CPU uses to operate the I/O units are contained in

ROM within each of the units. A disk drive, for example, actually *tells* the CPU how to read data from a disk and write data to a disk. Your computer, therefore, has what computer specialists call *device independent I/O*, a feature that saves the console unit from obsolescence when new peripherals are developed. It's due to device independent I/O that the choo-choo train peripherals can be replaced by more compact expansion box peripherals, and that expansion box peripherals can in turn be replaced by the yet more compact HEX-Bus peripherals.

The Power Supply

The TI-99/4A runs on three direct current voltages: +5, +12, and −5 volts. The role of the power supply unit is to convert 117 volts, alternating current, from your wall outlet into the three voltages required by the computer. The unit plugged into (or in earlier versions, mounted on) the wall outlet is the transformer for the supply. Other components (rectifiers, filter capacitors, and regulators) that develop the three direct current voltages are inside the console cabinet.

NOW THAT SOMETHING IS DELIVERED

Now that the screen shows the title display, and we understand some of the major concepts underlying the operation of the computer, we're ready to move on securely but rapidly. With the title screen showing, odds are greatly in favor of our connections being right and the computer's being completely in order. Now we can move on to the next chapter to check out the computer further by programming it. If you've never programmed a computer before, the next chapter will, in 30 minutes maximum, let you experience what you've missed (and maybe feared). "T minus 30 minutes" and your home computer is further checked out, and your uncertainties about programming are only memories and chuckles.

Chapter 3

How to Get Your TI-99/4A Up and Running

We've discussed, in Chapter 2, how you unpack and connect the various units of the TI-99/4A system. At this point, you should have all your cables correctly connected, any expansion peripherals installed, all units in the system turned on, and the title screen displayed.

In this chapter, we'll further check out the system by entering and running some small programs. As we check out the system, we'll use the experience to further familiarize ourselves with the operation of the equipment.

THE BUSY KEYBOARD

The keyboard on the TI-99/4A uses the QWERTY layout found on standard typewriters. That is, the first five keys in the top row of alphabetic keys are the letters QWERTY. If you touch-type, the location of letters and numbers on the 44-key keyboard, as shown in the Figure 3.1, should be familiar to you. This keyboard is, quite a bit "busier" than an ordinary typewriter keyboard. Let's look closely at the P key as an example.

Like most of the alphabetic keys, the P key has the character it represents printed in the middle of the top of the key. As on a typewriter, the letter P is printed in either upper or lowercase, depending on whether the shift or shiftlock (alphalock) key has been pressed down.

On the front surface of the P key, a double-quotation mark (") is printed. This indicates that if you hold down the function (FCTN) key at the lower right of the keyboard when you strike the P key, a quotation mark is produced. This one P key can thus produce three different symbols: a lowercase "p," an uppercase "P," and a double quotation mark.

Keys on the top row, similarly, have multiple uses. The numerals printed in the lower portion of the tops of the keys are produced if the SHIFT key is not pressed (regardless of the state of the alphalock key). The punctuation and symbol characters in the upper portion of the keys are produced if they are pressed while the SHIFT key is being held down. A third and even a fourth character may be produced from each key in this row if the function (FCTN) or control (CTRL) key is pressed when the key is struck.

The characters produced by top row keys when the control or function key is being held down are not identified on each key. Actually, the characters produced vary with the software you're using. The keyboard overlay strip packaged with the console, identifies characters produced when ten of the keys in this row are struck while the function key is pressed *while BASIC* is in control of your computer. While the function key is pressed, the leftmost key sends a *delete* command to BASIC, the key to its immediate right sends an *insert* command, and so on. As the overlay indicates by blank spaces, no character is produced by the zero key when the function key (matched with the white dot) is pressed, and no character is produced by any of the top row keys when the control key (matched by the red dot) is pressed.

The keys vary from program to program in the characters they produce. For characters produced when the function key is pressed, the Extended BASIC overlay labels the keys identically to the BASIC overlay. For characters produced when the control key is pressed, however, the Extended BASIC overlay labels all keys except the zero key. Although pressing the control key and any top row key while BASIC is running has no consequence, pressing the same combination of keys while Extended BASIC is running produces the "red-dot" character directly above it on the overlay. The key labels in the Terminal Emulator II program overlay are (except for QUIT) completely different from those of either BASIC program.

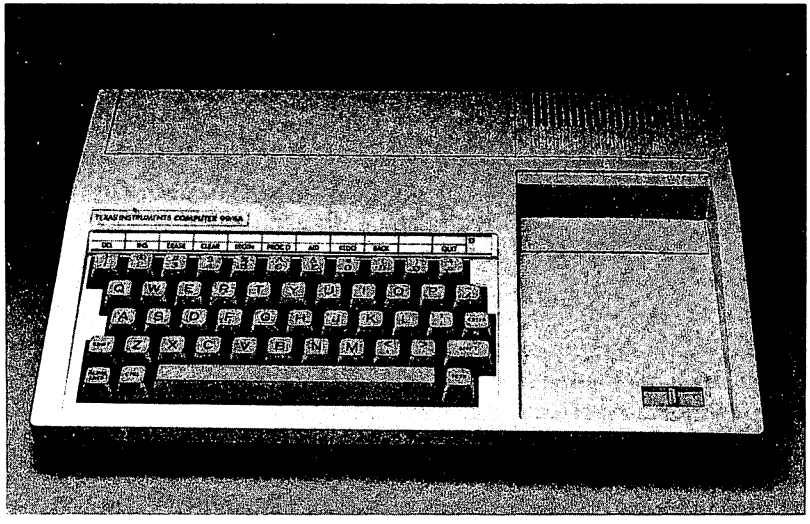

Figure 3.1 The TI-99/4A console, showing the keyboard.

Some of the keys on the TI-99/4A keyboard contain symbols that may seem a little strange (arrows, back-slanting apostrophes, and the like). However, the use of these keys is simple. We'll learn to use these keys as we need to in the next few chapters.

KEYBOARD ENTRY OF A LINE

Let's try a little work with the keyboard. With everything plugged in and turned on and the title screen showing, strike any key. The title screen goes away and you are presented with a menu. After you choose TI BASIC, the words TI BASIC READY appear on the third line from the bottom of the screen. On the last line of the screen is a *prompt* made from a greater-than sign. This prompt signals where characters you type in are displayed.

The TI-99/4A says it's ready, and we're ready, so let's get on with entering a line. Press the alphalock key at the lower left of the keyboard so that it latches down. Then carefully type the following characters, being careful to press the function key to produce the double quotation marks from the key.

```
10 PRINT "THIS"
```

If you make a mistake while you are typing the line, simply press the CLEAR key instead of the ENTER key and retype the line correctly. It's only when you press the ENTER key that the line is put into memory.

Now press the ENTER key. You have entered your first instruction into the memory of the home computer. The ENTER key tells the computer you have finished typing material on a particular instruction line—in this case a line you have numbered 10. After you press the ENTER key, your computer will move the black square called the *cursor* down one line so that it blinks at the left edge of the screen, indicating it's ready to accept another entry.

If you want to see what the line looks like in memory, enter the word **LIST**; this word is a command for the computer to display lines typed into program memory space.

You may be in for a surprise if you have never programmed before. With this instruction line in memory, you have also completed your first program. All that you need to do now is to tell the computer to follow the instruction, or in customary computer talk to *run the program*.

So, let's tell it. Type the word **RUN** and press the ENTER key.

The computer follows the instruction by changing the screen background color and printing the word THIS on the screen. Then it skips a line and prints * * DONE * * to tell us that it has finished all the instructions we have provided it.

Not to Be Mistaken

You are terribly luck if you haven't already discovered that typing lines perfectly before entering them can be difficult. Fortunately, there are several methods we can use to correct errors in a line already partially or completely typed.

Before Pressing Enter

If you spot an error before you press the ENTER key, you can use the arrow keys to move the cursor around on the line to pinpoint the character that needs to be corrected. The left arrow key, which quite naturally moves the cursor to the left, is produced by pressing the function key and striking the key labeled S. The right arrow key is the function key plus the key labeled D.

Let's correct a mistake of the sort that occurs when we strike a wrong key. Type in a new line 10 as follows, but do not press ENTER.

```
10 PRINT "THES"
```

Now, with the cursor just to the right of the second double quotation mark, press the left arrow key three times. Now the cursor blinks over the E character. Press the correct character, I, and notice that we now have the same line 10 that we entered correctly before. Press ENTER to put the line into memory. (There is no need to move the cursor to the end of the line; all characters typed on a line are stored in memory, regardless of where the cursor is on the line when it is entered.)

Next, suppose you have an extra letter in a line. Type in the incorrect line (without entering it) as follows:

```
10 PRINT "THEIS"
```

As above, use the left arrow key to position the cursor, this time positioning it over the extra letter E. Press the delete key (leftmost in the top row with the function key held down), and then press the ENTER key. Our single-instruction program is correct once more.

Now let's use the cursor to correct a missing letter. First type in the mistaken line as follows:

```
10 PRINT "TIS"
```

Next, position the cursor on the character following the missing letter, the I. Press the insert key (on the top row, second from the left, with the function key held down). Then press the missing letter H, and press the ENTER key. We now have a corrected line.

The insert key permits us to insert more than one character. To see this, let's do the same correction with a slight modification. Tap the H key twice this time, instead of once. Notice that we get the following line:

```
10 PRINT "THHIS"
```

By pressing the insert key, we have told the computer to insert characters at the cursor until we tell it to stop. And we must tell it to stop if we have to correct additional mistakes that require overstriking (replacement) rather than insertion. To tell the computer to stop inserting characters at the cursor and begin overstriking them, simply press one of the arrow keys.

After Pressing Enter

Often you will not notice an error until you have already pressed the ENTER key. Again there are several things you can do.

One method of correction is to retype the line. Suppose you have typed the following line and pressed the ENTER key before you noticed the error in the word **PRINT**.

```
10 PTINT "THIS"
```

If you retype the line correctly and then press the ENTER key again, the new version of line 10 will replace the old version in the memory of the computer. This is often the simplest and easiest way to correct an error.

If you accidentally type in a line that shouldn't be there at all, just type the number of the line and press enter. That will cause the entire line to be deleted from the program you are writing (but not from the screen). Using the line 10 we have been working with in memory, for example, type 10 and press the ENTER key. Line 10 is no longer in memory.

There is another process called *editing* that essentially recalls a line to the screen as if it had just now been typed, with the cursor positioned above the first text character of the line. To edit line 10, enter the following command.

```
EDIT 10
```

TI-99/4A BASIC responds by displaying the line for you. You may move the cursor with the arrow keys and use the insert and delete keys just as you did when you were correcting a line before pressing ENTER, with a single exception. You cannot change the line number (the cursor does not move over the numerals). When you have finished editing the text of a line, press ENTER. You may review the results of your editing by entering **LIST**, just as we discussed above.

A LITTLE, BUT NOT *TINY*, PROGRAM

You'll have to admit, once the elation wears off from writing a program without the pain of knowing that you were doing it, that one-line programs don't seem very challenging. Let's do a longer program.

Type in the following lines now, so that we can look at them in detail on the screen. Remember to press the ENTER key when you finish typing each line, and remember that you get capital letters without pressing the shift key, by having the alphalock key pressed down. If you make any mistakes in your typing, use any of the error-correcting features we've discussed to correct them.

```
10 INPUT " What is your name, please?" :N$
20 PRINT N$;" , do you write"
30 PRINT " computer programs (Y/N)?" ;
40 INPUT A$
50 IF A$ = " Y" THEN 80
60 IF A$ = " N" THEN 100
70 GOTO 20
80 PRINT " Good. We'll have some fun."
90 END
100 PRINT " Do not worry. You will learn easily."
110 END
5 CALL CLEAR
```

Notice that when you type line 20, which has exactly 40 characters (counting the > to the left of the line number), the cursor automatically moves to the next line after you type the last quotation mark. Even though the cursor is on another line, you must still press the enter key at the beginning of this line to tell the computer you have finished the line, because a program line can be much longer than a screen display line. In fact, you can enter a program line so long that it takes up *four* lines on the screen.

At this point you have your first "intelligent" multiple-line program in your computer. Let's look it over with our **LIST** command, but first let's unclutter the screen. To erase everything from the screen and place the cursor on the bottom line before we list the program, type in the following command.

```
CALL CLEAR
```

Press ENTER after you type the two words.

Now type the word **LIST** and press ENTER. Your program is now listed on the screen in numerical order. Even though you entered line 5 after the

other lines, it is placed first on the screen because it has the smallest line number.

The lines you have just typed in compose a computer program written in TI-99/4A BASIC, a surprisingly powerful version of the popular programming language. Your computer uses a relatively powerful dialect of BASIC that has much in common with the versions of BASIC used in other personal computers such as the Apple II, the ATARI 600/800, the Commodore 64, and Radio Shack TRS-80 Color Computer. If you learn BASIC on the TI-99/4A, it will not be difficult for you to adapt your knowledge to the version of BASIC used on other popular computers.

This program is a set of instructions you want the computer to follow. Each line contains an instruction, and each line begins with a line number. When the computer is told to follow, or *execute*, the instructions in a program, it begins with the instructions in the line with the smallest number. It executes the instruction in that line first; then it moves on to the line with the next larger number and executes the instruction in that line. It thus moves in numerical sequence through the program, from lowest line number to highest line number.

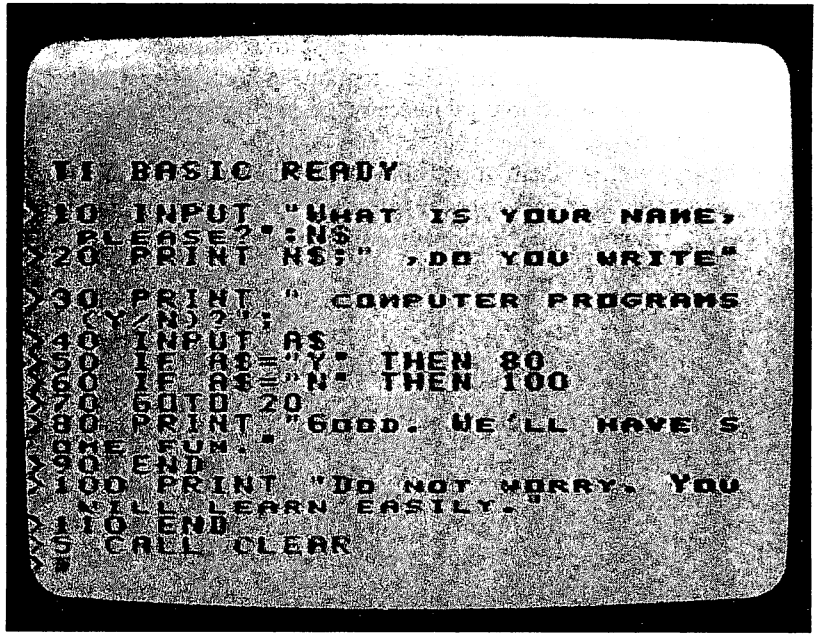

Figure 3.2 What your program looks like on the screen.

Do you remember how we get the computer to execute the program? Enter the command keyword **RUN** to tell the computer to follow the instructions in this program, which is now in its memory.

When you press the ENTER key after you type RUN, TI BASIC scans through the program to check each line for *syntax* errors. A syntax error is something in the way a line is formulated that prevents BASIC from trying to interpret it. If you have typed **PTINT** instead of **PRINT** in line 20, BASIC scans it after you enter RUN and displays the following message.

```
* INCORRECT STATEMENT IN 29
```

If BASIC finds no errors in the program, it tries to run it. If it is successful, your program produces the following sequence of events.

The screen clears, and the question *What is your name, please?* appears at the bottom of the screen. The cursor blinks beside the question mark. Type in your first name and press ENTER. If you type in Rumplestiltskin (without quotes) and press ENTER, the computer will reply, *Rumplestiltskin, can you write computer programs (Y/N)*, a question mark, and the cursor. Answer either Y or N for yes or no, and then press ENTER. If you enter N the computer responds *Do not worry. You'll learn easily.* If you enter Y, the computer responds *Good. We'll have some fun.* If you enter anything else, even lowercase y or n, the computer responds by asking the question once more. The program, as we'll discuss below, is designed to ask the question until it receives one of the two responses, Y or N, that it understands.

You have now entered and run a more challenging program. If it doesn't work as expected, look back over the lines you typed in and try to find the problem. When you have the program running correctly, we'll discuss the commands given to BASIC in each line.

A Look at the Lines

Chapters 6, 7 and 8 deal in detail with the BASIC language used by your computer. Let's be content here to get a general understanding of what happens in each line of the program and some idea of how lines are put together to make up a program.

The first line of the program does exactly what you did a moment ago before listing the program. Because the line number makes **CALL CLEAR** a step in this program, the command doesn't have the immediate effect that it did when you entered it without a line number. With the line number, it clears the screen and sets the cursor to the last line only when the program is run.

The next line of the program prints a message for you to enter a response, waits for your response, and stores your response in memory. The line is as follows:

```
10 INPUT "What is your name, please?" :N$
```

The keyword **INPUT** instructs the computer to wait for a response and to place whatever response you give it into a special place in memory. We don't care *exactly* where in memory the computer puts the response, because BASIC will find it for us automatically whenever we tell it to use the N$ we place after the colon as the last item in the line. We have, in effect, told BASIC to file the name we enter while the program is running in a *file folder* in memory called N$, and N$ is how we'll tell BASIC to get the name when we need it in another instruction.

The sentence in quotation marks is a *prompt* of our own making. We can write whatever we want to here, as long as what we write is surrounded by double quotation marks and followed by a colon.

The next three lines, 20, 30, and 40, display another prompt and call for another response. These lines are most interesting because they give a good insight into how computers are used to "magically" create business correspondence like form letters and monthly bills. These are the lines responsible for the program's addressing you directly by name. The lines are the following:

```
20 PRINT N$;" , can you write"
30 PRINT " computer programs (Y/N)?" ;
40 INPUT A$
```

Notice that the first of these three lines immediately calls for BASIC to get the name from the place it has stored it in memory in line 10. There are a couple of intricacies in the **PRINT** statements in lines 20 and 30. The semicolon in line 20 tells BASIC that it is printing two items which it must put together on a line: the name before the semicolon and the literal expression in quotes after the semicolon. Without the semicolon, BASIC might try to look for something it has filed under N$" . Similarly, the semicolon at the end of line 30 tells BASIC not to print the question mark produced by the **INPUT** statement in line 40 on a new line, but to put it as a continuation of the prompt printed by line 30. Because of the semicolon, the cursor also remains on the same line. Line 20 has no semicolon at the end, forcing the next displayed item to be on a new line (to leave room for names like Rumplestiltskin on the old line).

The **INPUT** statement in line 40 prints a question mark (it always does when there is no prompt sentence inside of double quotes) and tells BASIC to wait for your response on the keyboard. It files your response under the label A$ in memory.

Lines 50, 60, and 70 are most interesting because they give us good insight into computer decision-making. The lines are the following:

```
50 IFA$ = " Y" THEN 80
60 IFA$ = " N" THEN 100
70 GOTO 20
```

Line 50 tells BASIC what line to execute next if the response to the question "Can you write computer programs?" (the response it has filed by the label A$ in line 40) is the letter Y. Whenever Y is the answer, line 80 is going to be the next line executed. Line 60 does the same sort of thing whenever N has been answered: an N response tells BASIC that line 100 is the next line to be executed.

Line 70 is the line that tells BASIC what to do next if it doesn't understand the response that has been entered. In other words, if program control hasn't "gone to" line 80 or 100 with the recognition of a Y or N response, then program control "goes to" line 20 to ask the question again. Notice that the responses y and n, because they are not the same as Y or N, force program control back to line 20. We could provide for these lowercase responses to make our program more "user-friendly" by inserting the following pair of lines into the program.

```
61 IF A$ = " y" then 80
62 IF A$ = " n" then 100
```

Line 80 is what BASIC executes when a Y response is given. Line 90 tells BASIC that the program is over after printing the text in line 80.

```
80 PRINT " Good. We'll have some fun. "
90 END
```

Line 100 is executed when a N response has been given, and line 110 tells BASIC that the program is over after dispalying the text in line 100.

```
100 PRINT " Do not worry. You will learn easily."
110 END
```

These four lines demonstrate the use of the **END** statement. If the **END** statement in line 90 were not present in the program, line 100 would be executed, and therefore both computer responses would be displayed. The **END** statement in line 110, on the other hand, is included as a matter of

good programming form only. Deleting this line has no effect on how the program runs.

At this point you may be thinking that even this second sample program is too simple—that writing programs for a computer is far more complicated than this. It's true that there are some commands that are more difficult to understand than the ones we've used so far. And there are complex techniques used by professional programmers that can't be picked up easily in a weekend while keeping one eye on the football games and the other eye on the computer screen.

But it is also true that you can quickly and easily learn to write your own programs in this computer language called BASIC. The skills you learn are some of the same ones used by programmers who spend most of their waking hours at the keyboard of a computer. In addition, when you learn the commands in BASIC and how programs are written, you will be able to use programs you find in magazines and books, and you can adapt and personalize them as you see fit.

WHERE ALL GOOD PROGRAMS GO: THEY'RE SAVED

Before we get into more challenging commands and more mystifying programming techniques, we should prepare ourselves to solve a big problem we're going to be faced with.

Each program you have entered in this chapter is gone forever, when you enter another program to turn off the power to the system. The memory that stores them is volatile. Let's pause for a little while and explore saving and loading programs. When we start writing longer programs, we'll certainly want to know how to save them.

Chapter 4

Output to Cassette, Diskette, or Printer

If you have a program in the memory of the computer and want to store it for later use, in the language of computerese what you want to so is save it. That means that you want some way of keeping the program intact when the computer is turned off and the memory is erased.

Suppose you have typed in a program (one of those in Chapter 3, for example), and you'd like to show your skeptical friends the new programming talents you've acquired. But you won't see them for a week or so. Other than leaving the computer on all the time and using it for no other programs, what can you do? This chapter explains how to solve your problem by saving your program.

There are two electronic devices you can use for saving a program and later restoring it to memory: a cassette recorder or a disk unit. There is another device that is less often considered a storage medium: a printer. The cassette recorder and disk unit store information electronically in codes that make sense to the computer. The printer, on the other hand, provides what is called a *hard copy* of the program, which makes sense to you. A hard copy is the program transferred to paper by the printer; you can transfer it back to the home computer through the keyboard.

If you are using a cassette for storage, you need a TI cassette-recorder cable and a good quality audio tape cassette to insert into the cassette recorder. If you are using a disk unit, you need a 5 1/4-inch, soft-sectored diskette that has been prepared (formatted) for programs. Information on how to format a diskette is provided below. This chapter describes first the steps for saving a program to a cassette recorder, then to a disk unit, and finally to a printer.

SAVING TO CASSETTE

Before you can use a cassette recorder with the TI-99/4A, you must connect the recorder by cable to the D connector on the back of the computer. See Chapter 2 if your recorder is not connected.

If your computer is not already turned on and running BASIC, turn it on. The title screen appears.

Now, press any key. The program menu screen appears.

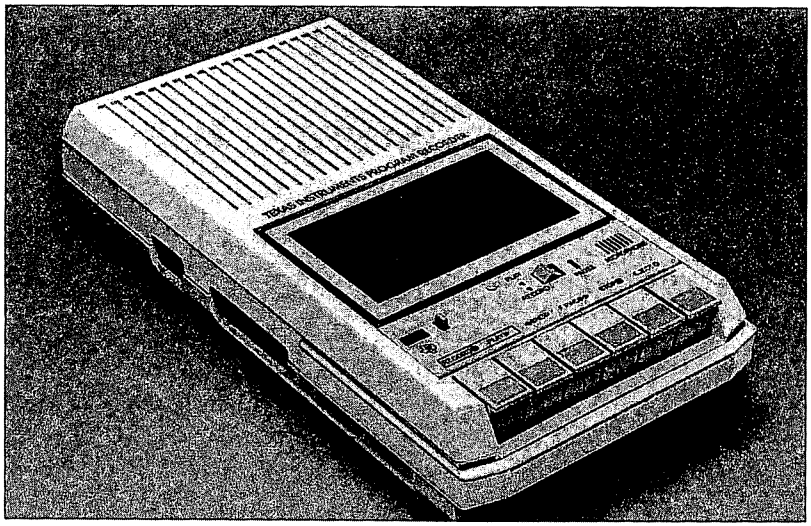

Figure 4.1 The TI Program Recorder

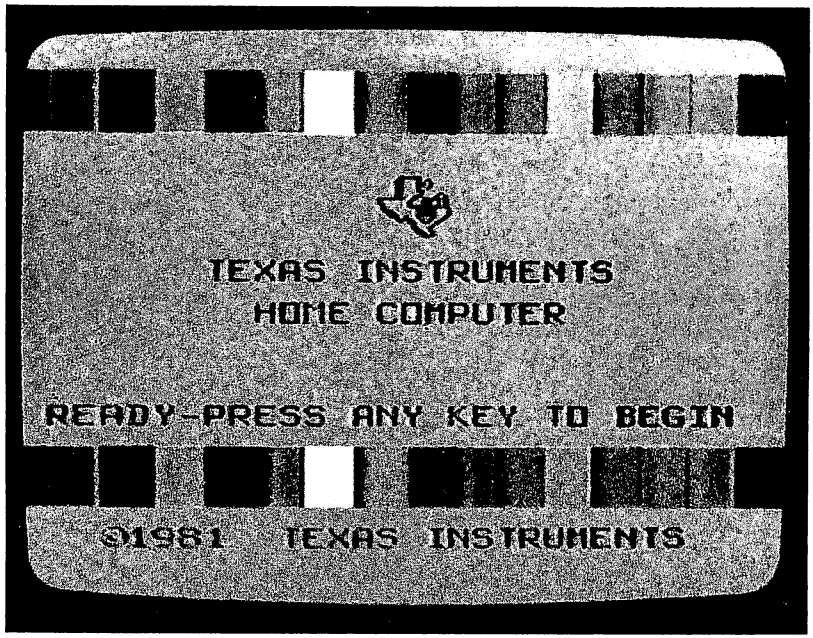

Figure 4.2 The TI-99/4A Title Screen.

Press 1 and ENTER, and your home computer will start running under the control of TI BASIC.

If you have just turned your computer on, enter a short program (such as the one-line program we wrote first in Chapter 3). If you haven't turned off your computer since you entered a program, you're all set to go.

Prepare the tape recorder by inserting the tape. If the recorder has a tone control, turn the control to maximum treble (highest tone). If the recorder does not have automatic recording-level circuitry, set the record-level control to the middle of its range.

STEP 1. Enter

SAVE CS1

The screen displays:

```
REWIND CASSETTE TAPE      CS1
THEN PRESS ENTER
```

Follow the directions and rewind the tape. Set your tape counter to 0. Then press ENTER.

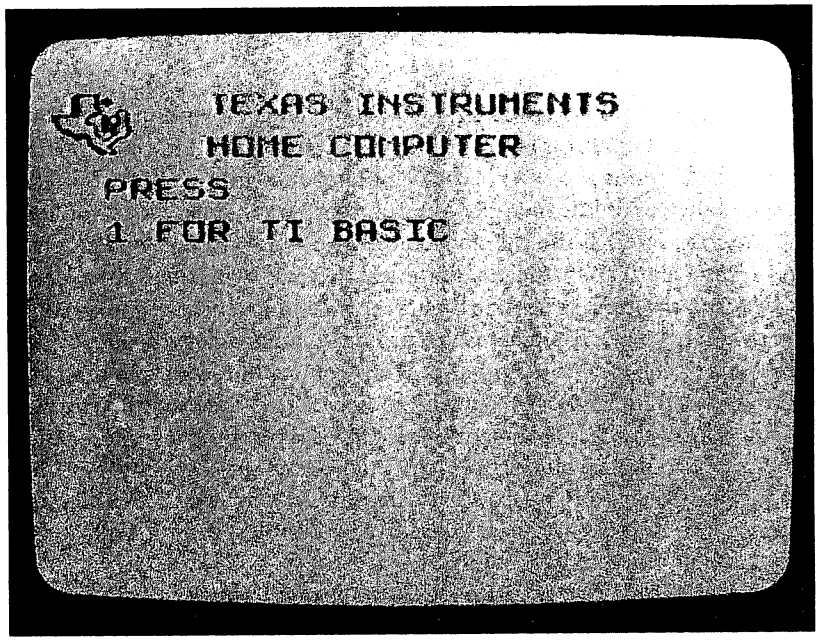

Figure 4.3 The TI-99/4A Menu Screen.

The scren displays:

```
PRESS CASSETTE RECORDER      CS1
THEN PRESS ENTER
```

STEP 2. Press RECORD and PLAY, and the recorder starts automatically. It is crucial that both RECORD and PLAY be pressed. The computer can't sense if only one of them is pressed. As with anything that is recorded on a tape player, if both RECORD and PLAY aren't used, nothing is recorded (in this case, saved). After your tape unit is set to record, press ENTER.

The screen displays:

```
RECORDING
```

When the program is saved, the screen displays:

```
PRESS CASSETTE STOP      CS1
THEN PRESS ENTER
```

After you have pressed ENTER, the screen displays:

```
CHECK TAPE (Y OR N)?
```

You have completed (hopefully) saving your first program. One important step remains, however. You should check to see that the program is really there on tape.

STEP 3. Before you do anything else, make a note of the numbers on your tape counter. For example, if the program starts where the counter indicates 5, and it ends at 12, write the beginning and ending numbers on the cassette label. This information is useful in two ways. First, when you load the program later, if the memory counter goes past 12 and the program isn't ready, you'll know there is a problem. Second, when you put another program on the tape, you'll know where this program ends without loading it. For example, we could save the current program again. Where on the tape should you save the program? Well, you could rewind the tape and load the program from the first time you saved it in order to find where it ends, and you could save it again there. Or, because you know the location on the tape counter, you can rewind the tape, reset the counter to zero and fastforward the tape to about 15 (a safe distance past where the program ends) to save a new program. Using the tape counter speeds up your saving and loading programs quite a bit.

Because strange things can happen when programs are saved, TI provided a way to check the program while it is still in memory. Why is this such a nice feature? Suppose you have just saved an important financial

program because you'll need it next week when your income tax return is audited. Although you're in a hurry, you decide to check the program anyway. To your horror you discover that the program, indeed, was not saved correctly. You get a new tape, save and check the program again and everything is fine. If you hadn't checked the program when you saved it, you wouldn't have known that the program wouldn't work until you tried to run it later—when the original was erased from the computer's memory. It is better to discover a problem while the original is still in memory. The crucial element here is to check the saved version with the program that's in memory. The time to check is while the original is right there in front of you.

Enter Y in response to the CHECK TAPE (Y/N) prompt.

The screen displays:

```
REWIND CASSETTE TAPE      CS1
THEN PRESS ENTER
```

After you have rewound the tape and pressed ENTER, the screen displays:

```
PRESS CASSETTE PLAY       CS1
THEN PRESS ENTER
```

When you have followed the instructions, the screen displays:

```
CHECKING
```

If everything goes well, and your program is on tape, the screen will contain the message:

```
DATA OK

PRESS CASSETTE STOP       CS1
THEN PRESS ENTER
```

You have finished saving and checking the program.

As long as the saved version matches the version in memory, the messages listed above appear. But what if there's a problem? If a problem has occurred during the **SAVE** procedure itself, one of the following three messages appears.

```
ERROR—NO DATA FOUND
PRESS R TO RECORD
PRESS C TO CHECK
PRESS E TO EXIT
```

or

```
ERROR IN DATA DETECTED
```

PRESS R TO RECORD
PRESS C TO CHECK
PRESS E TO EXIT

or

I/O ERROR 66

It may be difficult to determine exactly what the problem is, but you may want to **SAVE** the program on another section of the tape or on the other side. It's an even better idea to use another tape altogether. You might also have to use a different volume setting during your check of the tape.

DISKETTE STORAGE

Just as a cassette player must be connected to the computer, a disk unit must also be properly connected. Review Chapter 3 if you haven't yet connected a disk unit.

Remember that the power-up sequence—the order in which the computer and the expansion box (and any other peripherals if you are using a choo-choo train connection) are turned on—is very important with the TI-99/4A. When the drive is connected, turn it on first. Always make sure that the computer is turned on *after* the peripherals are.

Diskettes

Examine a diskette. The material you can see through the oval window of the diskette is where information is stored. Treat diskettes with care and *never* touch the plastic surface visible inside the oval window. Oil from your finger can prevent the computer from reading information stored there. Diskettes are sensitive and don't respond well to dust, dirt, chocolate chip cookie crumbs, or strawberry milkshakes. Because the platter inside the protective envelope is covered with a magnetic film, diskettes are also sensitive to magnetic fields. It is a good idea to try to keep them away from television sets and other electrical and electronic devices because they have transformers that generate a magnetic field.

The notch on the right side of the diskette is called the *write-protect* notch. If this notch is as it appears in the photograph below, the diskette is not write-protected. That is, the computer can store or change information on the diskette. If you have a diskette that contains a valuable program, and you don't want anyone to change it, you can place a piece of tape (most diskettes come with adhesive-backed tabs for this purpose) over the notch. When you write-protect a diskette in this manner, information cannot be put onto the diskette to overwrite information already there.

When you insert a diskette into the drive, always put it in with the window edge going first. Also, insert it so the label is to the right, away from the red light.

Notice the red light to the left of the drive. This light is on when the disk drive is reading or writing information.

Formatting a Diskette

In order to **SAVE** a program on diskette, you need a diskette that has been formatted. When you buy a diskette that has a program on it, that diskette had to be prepared before the program could be put on it. This preparation is called formatting and involves writing a sort of electronic map on the diskette. Later, when the computer wants to store information on the diskette, it uses the electronic map as a guide to tell it where to place the information.

Before you can use a new diskette to store programs or data, you must also use the format procedure. It's best to use blank diskettes (disks that don't have programs on them), because the formatting process erases anything that may be on the diskette.

To format a diskette, you must use the Disk Manager or Disk Manager II cartridge that comes with the drive.

To format a diskette, follow these steps.

STEP 1. Insert a diskette in the drive, and close the door.

STEP 2. Insert the Disk Manager cartridge into the cartridge connector.

STEP 3. From the menu screen, select 2 for Disk Manager.

SETP 4. After the Disk Manager title screen appears and you press any key, a menu appears. Enter 4 to set up a single-disk operation, and then enter 2 to bring up a menu of disk commands.

STEP 5. From the menu of disk commands, enter 4 to *initialize* a diskette.

STEP 6. Disk Manager will check for the diskette's presence and report the name of the diskette (if it has been previously formatted and named). Type in any name you choose— "DISK" is always a good name— and press ENTER in response to the next 3 questions about disk format. Disk Manager then formats the diskette (in a little less than a minute and a half). You are now ready to use the diskette with BASIC.

STEP 7. Return to the title screen by pressing QUIT, and proceed to enter BASIC.

Saving a Program

For the examples in saving to diskette, we can use the same program we used for saving to cassette tape. Any program will work. When you have a program typed or loaded in, follow these steps to save it.

STEP 1. Insert a formatted diskette into the drive and close the door securely.

STEP 2. Enter:

SAVE DSK1.PROGRAM

The red light on the drive comes on while the program is being saved. When the > prompt returns to the screen, the program has been saved under the name PROGRAM on disk 1. In a few seconds the red light on the disk drive goes off.

Although a file name is not used when you save programs to cassettes, it is impossible to **SAVE** a program on disk without a name. If you were to enter **SAVE " "** to try not to give a name, the following appears on the screen:

* I/O ERROR 64

OUTPUT TO A PRINTER

The last method of saving a program uses a printer to get a hard copy, or printed version, of the program. The TI impact printer (shown below), or any other standard serial or parallel printer, connects to the expansion box (or to older choo-choo train peripherals or a recently announced HEX-Bus peripheral) through an RS-232 interface.

An RS-232 interface converts processor signals into signals a printer can understand. Usually, RS-232 is a *serial* interface, but in TI peripherals it can mean *parallel* interface as well. The expansion box card includes a parallel interface as it comes from the box, and the HEX-Bus RS-232 interface includes one as an option. The essential difference between a serial and a parallel interface is in the way the electrical codes of characters are transferred. In a serial interface, they are sent as a series of ones and zeroes, one bit at a time. In a parallel interface, all seven or eight bits are sent at a time. A serial interface and printer have a much smaller connecting cable and a parallel printer is less expensive.

Connect your printer according to the directions packaged with the printer and your particular RS-232 unit.

With your printer connected, you can print one or more lines of information in a program, or you can have the lines of a program listed on

paper. When you are using a printer, the keyword you use to get a permanent copy of your program is still **LIST** but there is more to the command. An example in this chapter explains how to print a line-by-line listing of the program as well as how to print material from within a program.

Suppose you have a program like the one below. When you type the program in and enter the command **RUN** the program is executed. In the case of the program below, the phrase THIS IS A TEST appears on the screen, and then the * DONE message is displayed. Type in the following program to use the printer as a device to **LIST** your programs.

```
20 PRINT " THIS IS A TEST"
30 END
```

Now **LIST**, and when the listing is done, enter a **RUN** command. Your screen looks similar to this.

```
20 PRINT " THIS IS A TEST"
30 END
RUN
THIS IS A TEST

* * DONE * *
```

The computer sends its output to the screen. That is, when you enter **LIST** and **RUN**, both the listing and the text of the **PRINT** statement are

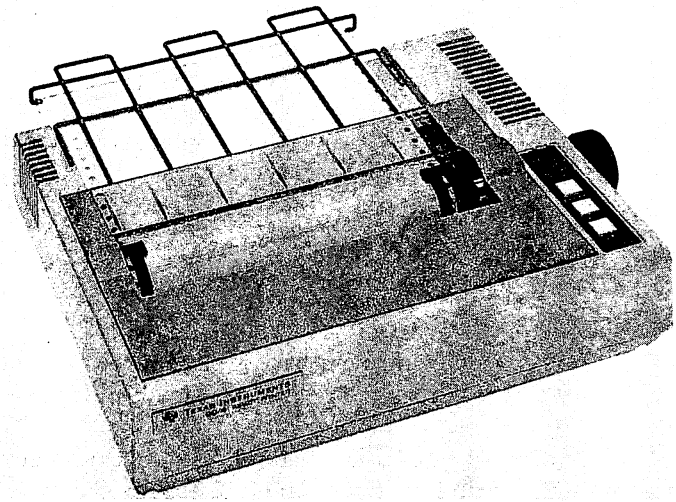

Figure 4.4 The TI Impact Printer.

displayed on the screen. However, you could have sent both to your printer.

To do this, you need to enter a few pieces of information by adding three lines to the program. As each new line is entered, we'll discuss its purpose. We'll assume that you have a parallel printer, but if you have a serial printer you can easily follow similar steps described in your manuals.

Enter a new line as follows.

```
5 OPEN #1:" PIO"
```

This line tells the computer that you want to **OPEN** a file numbered 1. (This is often called the logical-file or logical-unit number; it can be any number from 0 up to 255.) It also tells the computer that you want to access your parallel interface device with this file number.

Next we alter the **PRINT** statement in line 20 so that we print to the printer file rather than to the screen. We do this very simply by editing the line to read as follows:

```
20 PRINT #1:" THIS IS A TEST"
```

If a FILE ERROR message appears when this statement executes, you know that you have entered a file number other than the number specified in the **OPEN** statement.

After a file is opened and you have finished with it, the file should also be closed. If a file isn't closed in a longer program, an error message may result later on, although TI BASIC automatically closes the file when the program finishes its run. To close a file so that our programming is in good form, enter a new line as follows:

```
25 CLOSE #1
```

Now enter **CALL CLEAR** to clear your screen, and enter **LIST** to display the program with its new lines in numerical order.

```
 5 OPEN #1," PIO"
20 PRINT #1:" THIS IS A TEST"
25 CLOSE #1
30 END
```

By using the **LIST** command, you have a line-by-line copy of your program. Notice that the program line numbers are in the correct order now.

To get what's inside the quotation marks in the program to go to the printer, enter **RUN**. The printer types THIS IS A TEST.

Now we'll get a hard copy version of our program listing. Enter the following line:

LIST "PIO"

Now the listing of the program lines that earlier appeared on the screen is displayed on the printer. This is the way the printer is used to save programs. When the computer is turned off, the program and all the hard work that went into it are gone. With a printed listing, however, your creative work is not lost. You have a copy of the program on paper.

YOUR "OLD" PROGRAM IS SAVED NOW

In this chapter, we've dealt with ways to save programs to a cassette, a diskette, and paper. With your cassette or diskette, the program can be reloaded into memory, as we'll discuss in the next chapter. From a printer, we have the line-by-line **LIST** of the program; but to use the program again, you must type it in again on the keyboard. But saving to a printer largely protects the creative work of developing a program.

Now, in the next chapter, we'll see a peculiar and interesting thing about loading a program in TI BASIC. We're forced to call it an "old" program to get it back in the computer's memory.

Chapter 5

Loading Information from Cassette or Disk

We saw, in Chapter 4, how to **SAVE** programs using a cassette or diskette, and how to use a printer to provide us with hardcopy storage of programs. This chapter explains how to load programs using cassettes or diskettes. Loading transfers information stored on cassette or diskette once again into the memory.

CASSETTE STORAGE

The most difficult part of loading from a cassette (or a diskette, too, for that matter) is remembering the keyword to use. The keyword is *not* "load," and it doesn't sound like a command at all. The keyword is **OLD**.

Suppose you have a cassette that contains several programs, and you want to load the first program into memory. Perform the following steps, and you'll be ready to go with it.

STEP 1. Enter:

OLD CS1

The screen displays:

```
REWIND CASSETTE TAPE      CS1
THEN PRESS ENTER
```

Since you want to load the first program on tape, follow the directions. If you were interested in loading another program recorded after the first, you'd want to ignore the instruction and position the tape according to your tape counter.

STEP 2. Press ENTER

The screen displays:

```
PRESS CASSETTE PLAY       CS1
THEN PRESS ENTER
```

Of course, you must be careful not to press the record button as well.

STEP 3. Press ENTER

The screen displays:

READING

After the tape is read, the screen further displays:

DATA OK

PRESS CASSETTE STOP CS1
THEN PRESS ENTER

STEP 4. Press ENTER. The BASIC prompt > returns to the screen.

That's all there is to loading ("olding?") a program, unless, of course, something goes wrong. If something does go wrong, you'll get one of two messages quite different from DATA OK. One of the following messages will appear:

ERROR—NO DATA FOUND
PRESS R TO READ
PRESS E TO EXIT

or

I/O ERROR 56

If you're unlucky enough to receive one of these messages, you may have better luck if you try again to load the program. Make sure that you've set up the recorder as you did for checking the tape when you saved the program. The tone control should be set for highest frequency. The volume control should be set at approximately the level you set it for checking the tape. If all else fails, you can try reading again while setting the volume control at various settings.

If, on the other hand, you are lucky enough to avoid any error, you now have your program in memory. You can run it, list it, revise it, add to it, or otherwise work with it as if you had just entered it through the keyboard.

LOADING PROGRAMS FROM DISKETTE

In Chapter 4, we discussed the advantage a disk system provides in saving time (as well as programs). The same advantage is yours when you load programs from diskettes. There is a further advantage: saving and loading with diskettes is far more reliable than saving and loading with tape.

The process of loading programs from diskette is somewhat like using a cassette, only much faster.

There is really only one step to follow. Suppose you have inserted a disk that contains the program you want and have closed the disk drive door securely.

STEP 1. Enter:

OLD DSK1.PROGRAM

If the message I/O ERROR 56 appears, you've probably made a mistake in typing the name of the program. Otherwise, the BASIC > prompt returns to the screen to tell you that your program is loaded.

Some of the features of the disk drive that make it preferable to the cassette are evident when it comes to loading programs. For example, you need to know where on a tape a program is located to speed up the loading process when using a cassette. On a disk, however, you don't have to know where on the diskette a program is—the computer takes care of that for you. Another advantage of a disk drive is the ability to get a list of the programs on a diskette. This list, usually called a directory or catalog, lets you know what programs are on a diskette and approximately how much room each one will take up in the memory of the computer.

Suppose you have a diskette that hasn't been used in a while, and you don't remember what's on it. Insert your Disk Manager cartridge into the cartridge connector and perform the same series of steps you did to initialize the disk. This time, however, instead of entering 4 to select the disk command for initializing, press ENTER to automatically select CATALOG DISK. The screen will remind you of the name of the diskette and ask you where you want the listing of programs on the diskette. Pressing ENTER automatically selects the screen for catalog display, and you can then let the names of your programs roll up the screen.

If you have more than 12 programs on the disk, the screen won't hold them all. To stop them from scrolling past, simply press the space bar. To start them scrolling again, press the space bar once more.

Take just a minute to look at your directory. While the numbers and names might be different, your directory looks something like this:

```
                CATALOG DISK

DSK1 — DISKNAME = MYDISK
AVAILABLE = 218     USED = 115
FILENAME            SIZE        TYPE
--------------------  ------  ----------
INTEREST              10      PROGRAM
```

MEAN	1	PROGRAM
TEST	2	PROGRAM
TEMP	100	PROGRAM
YELOSHIP	1	PROGRAM

The information at the top of the display provides information about the whole diskette, and the information beneath the hyphens tells you about individual program files. The name of this disk is MYDISK. It contains 115 sectors of information that have been used by files, and it has 218 sectors that are available for use. A sector is, for the TI-99/4A, 256 bytes, or characters, of information.

The information provided about each of the files is a name, the number of sectors taken up by the file, and the type of information in the file. All of the programs we save are program files, although there are files of other sorts that you may save when you develop advanced programs.

PREFLIGHT

We've covered a great deal of information about the TI-99/4A home computer, and in the meantime, we've checked it out even further. You're probably comfortable with the keyboard and display now, and you may even be full of anticipation about what we might do next with our less-mysterious machine. From ground level, where we've been getting our bearings on the computer and its world, let's take off for a little checkout "flight."

You've glimpsed what BASIC can do for us in Chapter 3, and now that you can save, load, and perhaps print longer programs, you can begin to realize some of the real power of your computer. You're ready and your computer is ready, so let's fly with some BASIC programming power.

Chapter 6

Programming in BASIC

Now that we know how to save and load programs, we can look more thoroughly at BASIC. We assume that you have already read Chapter 3, so that you are familiar with methods for correcting and editing programs as you type them into the computer.

THE PURPOSE OF THIS CHAPTER (AND THE NEXT)

With over two hundred books currently in print that teach BASIC, we'd like to steer clear of such teaching. Instead, this chapter and the following one will introduce you to the concepts of BASIC and its use in the TI99/4A. In addition, you'll learn to enter and use BASIC programs written for your computer.

Several books have been published with listings of BASIC programs for the TI99/4A, and several magazines regularly publish programs that run on it (see Chapter 11). These sources of software are valuable because the software makes your computer a more powerful tool for work and enjoyment, and reading and entering these programs gives you more insight into the real nature of computer programming.

After reading the next two chapters, you should understand how BASIC programs work, and you should be able to take a TI99/4A BASIC program from a book or magazine, type it in, and use it. You will know enough BASIC to *debug* a program that isn't working as it should, and you'll even be able to make changes or enhancements that interest you.

It's possible for you to read through these two chapters without a computer, and if you do that you will get some worthwhile information about BASIC. However, we'd like you to read the BASIC chapters with a computer in front of you. As you read about new terms and techniques, use your computer to work with the related examples scattered throughout. We don't want you to try to memorize definitions and instructions. You don't need to be able to say off the top of your head what **GOSUB** 140 means (you only need to know where to look for that information when you need it). Rather, we would like you to work with BASIC enough so you're comfortable with it. After you work with BASIC for a while, it will become a second language for you, a language in which you think and communicate without memorizing a thing.

INTRODUCTION TO BASIC

There are several important differences between the language spoken by a computer and regular English. In English the meaning is usually clear even if a word or two is mispronounced or misspelled. BASIC and other computer languages are not so forgiving. You must say precisely what you intend to, and say it exactly as the computer expects.

Another difference is in the way punctuation is handled. Commas, semicolons, and periods usually clarify the meaning of written English, but the rules for their use vary. You could understand a letter from Uncle Harry even if he managed to write a whole page without a comma or period. In BASIC the punctuation marks are as important as the letters and numbers. Leaving out one comma, for example, can prevent an entire program from running.

Finally, English is a very rich language. There are usually several words that have similar meanings. A 1956 Chevrolet could appropriately be called a car, auto, automobile, vehicle, or even "wheels." BASIC is not so well endowed. Often there is only one way to say something. The words BASIC understands are called *keywords*. These keywords are used to make up commands or statements that make the computer do something.

There is a definite distinction beteween commands and statements. Commands (which some people call *imperative* or *immediate* statements) are the sentences you enter into your computer *without line numbers*. **LIST** and **RUN** as we used them in Chapter 3, are commands. Statements, on the other hand, are sentences we precede with line numbers. While commands are what you use in your interactions with the computer while you write or run programs, statements are lines in programs. Notice in the following lists, however, that commands and statements in BASIC sometimes look alike.

COMMANDS	STATEMENTS
LET A = 2	10 **LET** A = 2
LET B = 4	20 **LET** B = 4
PRINT A+B	30 **PRINT** A+B

If your computer is in front of you now, enter the sequence of commands. After you enter the first two commands, BASIC files the numbers 2 and 4, in much the same way it filed our names in the short program in Chapter 3. Then, after you enter the third command, BASIC retrieves the values, adds them, and prints the result. The sum, 6, appears beneath the **PRINT** keyword on the screen.

Commands are very useful, because they give you a way to try out your BASIC keywords and syntax without running whole programs. For example, if you can't remember the lowest frequency that BASIC accepts in a **CALL SOUND** statement, you can try it out in a command, as follows:

```
CALL SOUND(1000,100,0)
```

When you enter this command, BASIC displays * BAD SUBSCRIPT. The frequency of the sound, the middle number within parentheses, is too low. Now try the following pairs of commands:

```
CALL SOUND(1000,110,0)
CALL SOUND(1000,109,0)
```

In much less time than you could have looked up the number, BASIC commands have let you determine it firsthand.

While commands are processed by BASIC in the order you enter them, statements are always processed according to line numbers. When a computer is told to **RUN** a program, it begins with the statement that has the lowest line number and follows the instructions given in that statement. It then proceeds through the program until all the statments have been executed. In the three lines of statements listed above, when BASIC reaches line 30 it will do the arithmetic requested (2+4) and print the result on the screen.

Enter these three statements into your computer. You'll probably recall from Chapter 3 that these three statements make up a simple program, a set of instructions that tell your computer what to do. When the three-statement program is typed in, type **RUN** and press the ENTER key. If the computer responds by printing 6, both you and your computer are off and running.

SOME IMPORTANT COMMANDS

Several of the keywords in BASIC are used normally (and maybe *exclusively*) to tell the computer how to manage its work. They make up a *job-control* language that allows you to exercise control over the writing, editing, and running of programs. Let's look at these keywords and the jobs they can do for us.

BYE This command consists only of the keyword. It causes the TI99/4A to terminate its execution of BASIC and return to the title screen.

BREAK It's one of the sadder facts of computing life that BASIC programs we write sometimes don't work, but it's a happier fact of life that we can interrupt them while they are running in order to discover the

trouble. The **BREAK** command, which sets *breakpoints* at which a program is interrupted, is one of our major tools in troubleshooting programs.

The **BREAK** command tells BASIC to return execution to command level when it encounters specified lines in the current program. The line for which the breakpoint is set is not executed; if the program is continued after the breakpoint, that line is the first one executed. Breakpoints are specified as a list of line numbers following the keyword **BREAK**. The command **BREAK** 100,200,300 causes execution to stop at lines 100, 200, and 300, provided that lines with these numbers are actually in the program.

The **BREAK** keyword may be used within program statements, with or without a line-number list. Without at least one number following the keyword, **BREAK** causes execution to stop at the line on which it occurs.

Once a breakpoint is encountered and program execution is stopped, the breakpoint is reset. Execution will not stop at that particular line in the future unless the breakpoint is set once more. To continue executing the program after a breakpoint has been reached (and values have been examined, etc.), use the **CONTINUE** command. To cancel a breakpoint before a specified line is executed, use the **UNBREAK** command.

Breakpoints affect standard-character code redefinition (see **CALL CHAR** in Chapter 7). If a standard character code (say 32 for an ASCII space) has been assigned to a different character for display before the breakpoint is encountered, it is reassigned the standard character of space when the breakpoint is encountered.

CONTINUE This command causes a BASIC program stopped by a breakpoint to continue execution. The **CONTINUE** command cannot be used if a line is edited, added, or deleted while a program is running.

Because the break in program execution caused by pressing the CLEAR key functions as a breakpoint does, **CONTINUE** causes program execution halted in this way to be resumed, provided that no line has been edited.

DELETE If you want to delete a program or a data file that is on diskette, use this command. The command **DELETE** DSK1.DATA deletes the disk file called DATA on disk drive 1. Cassette-tape files cannot be deleted with this command (they must simply be recorded over with new programs, bulk-erased, etc.).

If you want, on the other hand, to delete a block of lines in a program, as you can in many other versions of BASIC, we've no good news for you here. You must type in the line number of each line you want to delete, following each line number by ENTER.

EDIT This command redisplays a previously entered line on the screen so that it can be altered through the use of arrow, delete, and insert keys.

The keyword must be followed by the exact line number of the line to be edited, or the * BAD LINE NUMBER message is displayed. The syntax for the command is, for example, EDIT 100.

Editing may also be evoked by typing the number of a line followed by an up or down arrow key.

Regardless of how editing is evoked, it can be continued with either the next higher or lower numbered line by pressing the down arrow key or up arrow key. This feature forms an alternative to the **LIST** command in which listing cannot be temporarily halted at the keyboard. Pressing and holding the down arrow key produces a slow scrolling of lines that halts whenever you remove your finger from the key.

LIST This command tells the computer that you wish the lines of the current program to be displayed on the video display. This is usually called a *listing* of the program. **LIST**, the keyword used by itself, not followed by any numbers, causes the entire program to be listed. If there are no more than 23 lines (including the *wrapped* second, third, and fourth lines in any long numbered line), the entire program will be shown on the screen. If your program is more than 23 lines long, the computer can't get all of it on the screen at once. **LIST** will cause the lines to fly by on the screen, and the final 23 will be the only ones you can actually read. For this reason, using the **EDIT** command (described above), then the arrow keys to scroll the program line-by-line, is a satisfactory alternative to fly-by listing or listing in line-number ranges. If your lines are numbered at constant intervals, the **NUMBER** command can also be used for more satisfactory listing.

The **LIST** keyword followed by a number lists the line of that number. If there is no line with that number, BASIC lists the next higher numbered line. If the number is greater than the highest numbered line in the program, BASIC lists the last line of the program. **LIST** 30 causes line 30, if there is one in the program, to be listed. If there is no line 30, but there is a line 40 (or any line in the program with the next higher number), BASIC lists line 40 (or whatever the next line is numbered). If line 25 is the last line in the program, BASIC lists line 25.

The keyword **LIST**, followed by a pair of numbers specifying a range of line numbers, will cause BASIC to list all lines with numbers within the range. The syntax (punctuation) BASIC recognizes is two numbers separated by a hyphen. If the range of numbers in the command is below the range of numbers in the program, the first line of the program is listed; if it is above the range of lines in the program, the last line is listed. If the range of numbers in the command lies between two program lines, the next higher numbered line is listed.

Either (but not both) of the numbers that specify a range for the **LIST** command may be assigned *by default* to be the first or the last line of the current program. For a program with a first line number of 30 and a last line number of 300, the command **LIST** -50 causes BASIC to list lines 30 through 50. **LIST** 250- causes BASIC to list lines 250 through 300.

NEW There are times when you want to stop what you are doing and start over. One way to accomplish that is to pull the plug on the computer. Another way is to use the command **NEW**. **NEW** erases any program (or filed values) currently in the computer's memory. You can then start fresh. Before you use **NEW**, be sure you really want to erase everything in memory. Once it is done, you can't undo it.

NUMBER This command causes BASIC to provide line numbers for you. If you are initially entering a program, entering the keyword **NUMBER** (or simply **NUM**) displays the line number 100 for you to enter text beside. Once line 100 (with the text you type in) is entered, the line number 110 appears, and so on until you have finished program entry. When you have finished, simply press ENTER in response to the line number that has appeared, and BASIC stops automatic production of line numbers and returns to the command line with the > prompt.

If you want to begin automatically numbering lines at a specific number, place a space and the number you want to begin with after the keyword. The command **NUM** 1000 renumbers the first line of the program to be 1000, the next to be 1010, and so on. If you want the automatic increment to be something other than 10 (1000, 1010, 1020, and so on), place a comma and second number representing the increment you want after the first number. The command **NUM** 1000,1 renumbers the first line 1000, the next 1001, and so on. If you want to have your own increment and still begin with the default line, 100, place a space, a comma, and the increment number after the keyword. The command **RES** ,1 produces an initial line with the number 100, a second line with the number 101, and so on.

If you are automatically numbering lines, and BASIC produces the number of a line already entered into a program, the line is displayed. It may be edited, and the edited version replaces the original version in memory when ENTER is pressed. Because there is text with the line, ENTER does not cause you to exit automatic line numbering. Pressing CLEAR while a previously stored line is displaying, however, lets you leave automatic line numbering and return to the > command prompt without altering the line you have cleared from the screen within memory. Pressing ERASE does away with everything but the line number, giving you the opportunity to (1) retype the entire text of the line, (2) stop automatic numbering while leaving the line intact by pressing ENTER if you don't

have any text on the line, or (3) stop automatic numbering while leaving the number intact by pressing CLEAR.

If the lines of a program already in memory are numbered on constant increments, **NUMBER** or **NUM** is (like **EDIT**) a convenient way to avoid the fly-by characteristics of the **LIST** command when you want to review program lines.

OLD This command, corresponding to the keyword **LOAD** in most other versions of BASIC, loads programs from diskette or cassette tape. (Review loading instructions in Chapter 5 if necessary.)

RESEQUENCE If you have lines that you want to renumber (so they are numbered at constant increments, or so you can insert lines without retyping lines with adjacent numbers, this is your "prettying-up" and labor-saving command. The keyword **RESEQUENCE** (or simply **RES**) may be used to renumber all lines of a program so that numbering begins at 100 and continues in increments of 10.

If you want renumbering to begin with a number other than 100, place a space and the initial number after the keyword. The command **RES** 1000 renumbers the first line to be 1000, the next to be 1010, and so on. If you want the increment to be other than 10, place a comma after the initial number and then type the new increment. The command **RES** 1000,5 renumbers the first line of the program to be line 1000, the next line to be 1005, and so on. If you want the first line to be 100, but you also want the increment to be other than 10, follow the keyword with a space, then a comma, and then the increment. **RES** ,1 renumbers the first line to be 100, the next to be 101, and so on.

RUN We've already used this command several items. When you give the **RUN** command, it tells the computer you want it to execute the instructions contained in the program currently in the computer's memory.

Normally, **RUN** tells the computer to begin with the instructions in the lines with the lowest number. You can also use **RUN** with a number that specifies where it's to begin. If you tell the computer **RUN** 70, the computer will start executing instructions beginning at line 70.

SAVE This command saves programs onto diskette or cassette. (Review the instructions for saving programs in Chapter 4 if necessary.)

TRACE This command, like the breakpoint commands, is one of those happy tools that helps us fix "broken" BASIC programs. The **TRACE** command tells BASIC to display the line number of every BASIC statement at the left of the screen before it executes the statement. With TRACE, we can see the steps being taken by the program as it is running.

If the program produces a displayed output from a line, the displayed output is listed immediately to the right of the line number. Lines are "wrapped" to accomodate the space taken up by the line number produced by **TRACE**.

Tracing through the programs is terminated when an **UNTRACE** command is entered.

Like the breakpoint keywords, **TRACE** and **UNTRACE** may be used with syntax identical to commands in program statements.

UNBREAK Although this command keyword may seem to be TI's unique contribution to either the BASIC or English language, it's really just a word that TI BASIC recognizes as an instruction to do away with breakpoints that have been sent (see **BREAK**, above). If **UNBREAK** is not followed by a list of line numbers separated by commas, all breakpoints are cleared. If a space and a list of line numbers follows the keyword, only the breakpoints set for the lines with numbers in the list are cleared.

The **UNBREAK** keyword, like **BREAK**, can be used as a program statement, as well as a command.

UNTRACE Like **UNBREAK**, **UNTRACE** hasn't made it into general usage in either BASIC or English, but it is another word that TI BASIC understands. The **UNTRACE** command turns off the line-execution tracing feature. If you trace through a portion of a program to your satisfaction and want to turn the tracing off from the keyboard, press CLEAR to stop the program, enter **UNTRACE** to stop tracing, and enter **CONTINUE** to resume program execution without the line numbers.

The **UNTRACE** keyword may also be used in program statements.

THE SPECIAL-FUNCTION KEYS

The special-function keys, which are in most cases the number keys with the FUNCTION key held down, are used for entering, editing, and running BASIC programs. You've used these keys as required in our sample programs, and we've seen some of the uses of these keys in our survey of important BASIC commands. Before you get into any longer programs, you'll want to take a more careful look at all of them to see exactly what each one does.

Remember to place your keyboard overlay strip in the slot above the keyboard so that the special function keys are labeled.

Five of the function keys are not used in writing and running TI BASIC programs. There are BEGIN, PROCEED, AID, REDO, and BACK. These keys are used in applications programs, such as the TI Disk Manager or Extended BASIC, for a variety of special purposes. In Extended BASIC, for example, REDO copies from a line in memory to a new line, providing you

with a tremendous amount of help in entering highly repetitious program lines.

ARROW KEYS The left and right arrow keys move the cursor backward and forward on a line. The up and down arrow keys are used following a line number to initiate line editing and to continue editing with a previous or subsequent line of a program. The arrow keys are the only special-function keys formed by holding down the FUNCTION key and striking alphabetic characters (S, D, E, and X); all others are formed by holding down the FUNCTION key and pressing a number key on the top row of the keyboard.

CLEAR This key has several important uses.

When you are typing a command, CLEAR erases the line and provides you with a new > cursor.

When you are initially typing a program line which you have provided the number for, CLEAR erases everything you have typed and gives you a new cursor. If you already have a line with that number in memory, it is unaffected.

When you are editing a line, it terminates your editing without changing the line in memory.

When you are automatically numbering lines, CLEAR terminates the automatic numbering without entering the current line. If the current line is already in memory from prior entry, it terminates automatic numbering without changing the line in memory.

Also, when you are running a program, it causes program execution to be interrupted as if a breakpoint were encountered. The number of the line that would have been executed next is displayed on the screen. Execution may be resumed with the **CONTINUE** command.

DELETE The delete key erases the character under the cursor and closes up the space that the deleted character occupied.

ERASE When you are entering a new line, ERASE clears the line and displays a new command cursor, just as CLEAR does.

When you are editing a line or BASIC is automatically numbering lines, ERASE clears all of the text of the line but not the line number.

INSERT When you want to insert characters into a line at the cursor rather than replace them, press INSERT. After you do, each key you strike causes both the cursor and the character under it to move to the right. The new character appears in the space created to the left of the cursor.

QUIT This key causes the computer to leave BASIC and display the title screen. All programs and data are erased when you quit BASIC, so be sure to save anything you want to work with later on cassette or diskette.

PRINT STATEMENTS

Now that you've gotten the hang of the commands and special keys, we'll pick up the pace a bit. In the following sections, we'll present and explain additional keywords and programming rules. Generally the explanation will include a short demonstration program. If this is your first adventure in programming, we advise you to read the explanations carefully and **RUN** each of the sample programs.

A print statement causes the computer to print out or display whatever follows **PRINT**. You have used it several times already. If material following **PRINT** is enclosed in quotation marks, the computer will print it exactly as typed. The material inside the quotation marks is called a *string* or a *literal string*. If you use the **PRINT** keyword in a command, you can see that the command **PRINT** "THIS IS A TEST" causes the computer to print the string THIS IS A TEST on the screen.

If there are no quotation marks around the material to be printed, the computer behaves a little differently. Consider the program below:

```
10 INPUT THIS$
20 PRINT THIS$
```

Enter and run the program. Line 10 tells the computer that you want to enter a variable (what we've called a filed name before now) called THIS$ from the keyboard. Line 20 does not print the name of the variable (THIS$). Instead, it prints what we have entered and BASIC has filed away under the name of the variable. It prints whatever we type on the keyboard. Thus, whether you use quotation marks or not will make a big difference in what the computer does.

There are three punctuation marks (semicolon, comma, and colon) we can place inside of **PRINT** statements in order to make the text displayed by BASIC to look the way we want it. Please add the following lines to the two lines you typed in to see how these punctuation marks work with PRINT.

```
30 PRINT " THIS IS" ;THIS$
40 PRINT " THIS IS " ;THIS$
50 PRINT " THIS IS" ,THIS$
60 PRINT " THIS IS" :THIS$
```

Note that the only difference between line 30 and line 40 is the space between the S in IS and the quotation mark. Now run the program. Now enter the word ENJOYABLE in response to the question mark displayed by the **INPUT** statement. Your display should look like the following:

```
ENJOYABLE
THIS ISENJOYABLE
THIS IS ENJOYABLE
THIS IS          ENJOYABLE
THIS IS
ENJOYABLE
```

The ENJOYABLE in the first line comes from line 20. Then lines 30 and 40 produced almost identical displays. The literal string THIS IS is printed first by both lines. Then the value of the variable THIS$ is printed. In line 40 a space inside the quotation marks makes the line neater and easier to read because the word ENJOYABLE is not directly next to the S in IS. The semicolon that follows the literal string tells the computer to print the next item immediately after the last item printed. If you use a semicolon to separate two items that are to be printed, you will need to add any spaces yourself (inside the quotation marks).

Line 50 produces a different result, with seven spaces appearing between THIS IS and ENJOYABLE. The comma tells BASIC to print the expression following it in the second field on the screen. There are two 14-character fields in each line, one beginning at column 1 and another at column 15. If, when BASIC finds the comma, the cursor on the current display line is not past column 15, the item following the comma is displayed there. If the cursor is past column 15, the item is printed in the first column of the next line. Try the following commmand.

```
PRINT 1,2,3,4,5,6,7,8,9,0
```

Line 50 is used to break items in a **PRINT** statement into separate lines. Everything after a colon goes on the next line. Try the following command.

```
PRINT 1:2:3:4:5:6:7:8:9:0
```

Semicolons tell BASIC to have no respect for the end of a line. If you tell the computer to print material that is too much to fit on a line, the computer will type all it can on one line and put the rest on the next line. Use the **NEW** command to clear the ENJOYABLE program, and then enter the following program.

```
 5 FOR X = 1 TO 20
10 PRINT " AND " ;
```

```
20 PRINT " YET " ;
30 PRINT " ANOTHER " ;
40 NEXT X
```

Run the program, and you will see that it fills up 12 lines on the display. The three literal strings AND, YET, and ANOTHER are printed end to end until the program loop has been executed 20 times. Now add the following lines:

```
50 PRINT
60 PRINT
70 FOR B = 1 TO 7
80 PRINT " EVEN" ,
90 PRINT " MORE" ,
100 NEXT B
```

Now run the expanded program. You get the same letters at the top of the screen. Then lines 50 and 60 give us a blank line (50 gets us off the seventh line, 60 skips one line). Then the second **FOR-NEXT** loop prints EVEN MORE on the screen in separated columns seven times. The comma on lines 80 and 90 causes the computer to space over and print material at the beginning of each of the two print zones.

You may want to experiment with **PRINT** instructions and commas, colons, and semicolons further to get familiar with how they work.

A final note on **PRINT**. If you tell the computer to print something and don't put a comma or semicolon after it (for instance, **PRINT** "HELLO") the computer will automatically go to the beginning of the next line when it encounters the next **PRINT** instruction.

TI BASIC recognizes a **TAB** keyword that prints the number of spaces you specify beside it. Here is a demonstration program.

```
20 FOR X = 1 TO 20
30 PRINT TAB(X);" X"
40 NEXT X
```

Be sure to type **NEW** and press RETURN before typing in this program. When you have it typed correctly, run the program. You should get 12 lines of material on the screen, which looks like Figure 6.1.

Line 20 begins a "**FOR-NEXT** loop", something we'll look into in detail later, while line 40 is the end of the loop. In essence, lines 20 and 40 will cause line 30 to be executed twelve times. Line 30, with the keyword and number TAB(X) tells BASIC to begin printing at the column numbered by the value of X. While X progresses from 1 to 20 to print line 20 twenty times, the column number of the tab progresses.

GETTING INFORMATION INTO THE COMPUTER

There are three major ways TI BASIC allows you to assign *values* to *variables*—the process we've referred to as *filing information in memory by label*. We've explored one of these ways already. You can enter characters and numbers into storage by using **INPUT** statements. Let's look at the details of the **INPUT** statement, and then we'll review the two other statements that assign variables from inside a program: the **LET** statement and the **DATA** statement.

Input

When the computer comes to an **INPUT** statement, it stops and waits for the operator to provide it with some information. This information may be in one of two forms: it may be either a number or a *string*. More technically, the **INPUT** statement includes the name of either a numeric or a string variable: this name is the label your input is filed under, as we've discussed above.

The letter X (without quotes) in our last example is a numeric variable. It is a number that starts out as 1, then is incremented to 2, and continues being incremented until it reaches 20.

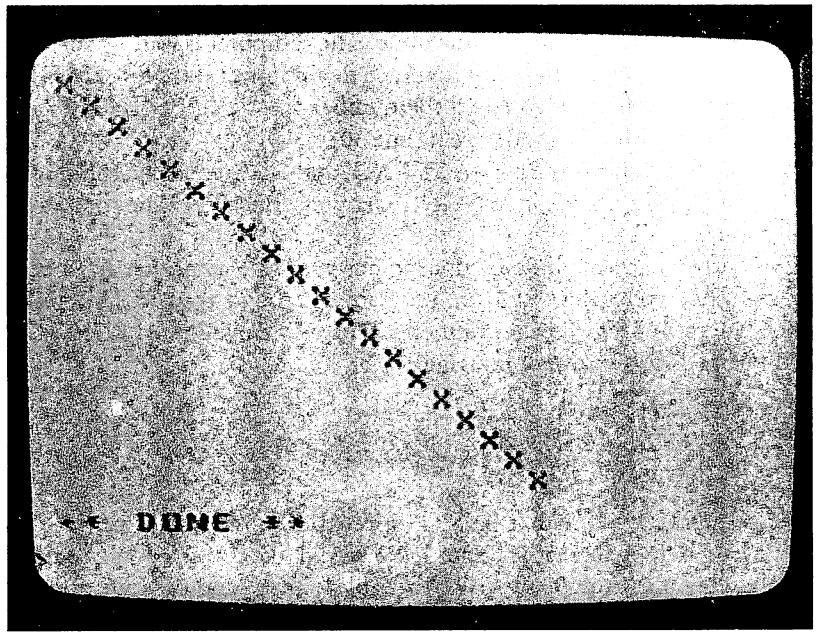

Figure 6.1 Using the **TAB** keyword.

In our ENJOYABLE program, we used the statement **INPUT** THIS$. The $ at the end of the variable name indicated that we were able to enter alphabetic, as well as numeric, information; in other words, the $ identified the variable as a string variable. If the variable name does not end with a $, BASIC assumes it is dealing with a numeric rather than a string variable.

If **INPUT** is followed by a numeric variable name (for example **INPUT** B or **INPUT** BALANCE), the computer expects you to type in a numeric value and press ENTER. If **INPUT** is followed by a string variable name (for example, **INPUT** B$ or **INPUT** NAME$) the computer expects you to type in a string (don't enclose it in quotation marks) and press ENTER. If you give a string value when the computer expects a numeric value, BASIC will reject your input. Here is a short example program that demonstrates the difference between numeric and string variables.

```
20 INPUT " TYPE IN A NUMBER" :A
30 PRINT A+2
40 PRINT " TYPE IN A STRING"
50 INPUT A$
60 PRINT A$
```

Run this program. The screen will display TYPE IN A NUMBER on one line, and a ? will appear on the next line. The computer is waiting for you to type in a numeric variable. If you type in a number and press ENTER, the computer will set A to equal the number you typed in. Then it will print the value with two added to it (line 30).

Line 40 tells you to TYPE IN A STRING, and line 50 tells the computer to look for a string. Type in TESTING and press RETURN. The computer will print TESTING.

Now what happens if your input is not what the computer has been programmed to expect? Run the program again and type in TESTING when the computer expects a numeric value. When you press RETURN, BASIC prints the following message:

```
* WARNING:
  INPUT ERROR IN 20
TRY AGAIN:
```

This message means that BASIC expected a number, that it did not expect you to type in a string, and that it would like you to try again. The problem is a mismatch between the variable name A and the type of data you entered, TESTING. Enter the number 34, and the program will continue. The computer accepts the number when you press RETURN and

prints it on the screen. It goes on to lines 40 and 50. Now it is looking for a string variable. Type in 34 and press ENTER. Surprised by the results? The computer accepted 34 as a string with no problems and printed 34. That means A equals 34, and so does A$. There is a difference in the two variables, however. Add the following two lines to the program to see the difference.

```
70 PRINT A+12
80 PRINT A$+12
```

Run the program and give both A and A$ the value of 34. When the computer gets to line 70, it computes 34+12 and prints 46. When it reaches line 80, it prints the following:

```
* STRING-NUMBER MISMATCH
```

You cannot add a string variable to a numeric variable. Thus, even though A$ equals 34, you will not be permitted to use it in math operations. To the computer, the 34 value of A$ is simply two characters—not a number. The variable A can be manipulated mathematically, while the variable A$ cannot.

Up to this point we have used simple **INPUT** statements in which the value of one variable was requested. An **INPUT** statement can be used to assign values to more than one variable. INPUT A,B,C causes the computer to print a ? on the screen. When you type in 44,55,66 and press RETURN, the computer accepts all three values at once. You can even mix numeric and string variables (**INPUT** A$, B, C$) but you must be careful to enter the values in the correct order (that is, in this example, string, number, string) and separate each with a comma. Also, you must provide all the values requested by the **INPUT** statement in your response. If you enter only 55 and 66 in response to the request above, BASIC notifies you of an input error and tells you the line number of the **INPUT** statement.

Notice that the **INPUT** statement always requires that you press ENTER after you have typed a character at the keyboard. Other dialects of BASIC, including Extended BASIC, have keywords that permit characters to be entered without pressing ENTER.

LET Statements

The **LET** keyword tells the computer what value to associate with a particular variable name. For example, **LET** B = 22 tells the computer that until further notice the variable named B will equal (have the value of) 22.

The general form of a **LET** statement is as follows:

LET variable name = value

If you assign a value to B (or B$) early in a program (for example, 10 **LET** B = 22) and then assign another value to B later (for example, 30 **LET** B = 66), the value of B will be the last one assigned (66, in this example).

Up to this point, the value assigned with the **LET** statement has been a simple numeric value (22) or a literal string value (HELLO). **LET** is much more versatile, however. **LET** can use another variable to assign the value to a new variable (**LET** B = A or **LET** B$ = A$), and it can use an *expression* (**LET** B = 2+5). These **LET** instructions are also acceptable.

```
LET B=B+2
LET B=B+A
```

The only restriction in the above examples is the requirement that any variable name on the right side of the equal sign must have been defined previously in the program. A program with line 30 as **LET** B = B+A is fine if an earlier line defined the value of A (for instance, 10 **LET** A = 95).

We should note here that the computer can use variable names that are more complex than A or B$. Variables can have names up to 15 characters long, inlcuding the $ that ends a string variable. Variable names must begin with an alphabetic letter, an *at* sign (@), a left or right bracket ([or]), a backslash (\), or an underline (_). Characters within a name may be letters, numbers, at signs, or underlines.

However, there are two additional limitations on the way you name your variables. TI BASIC has a number of keywords it treats as instructions. You have used keywords such as **PRINT**, **LET**, and so forth already. You cannot use any keyword to name a variable. **ABS**, for example is not an acceptable name since it is a keyword meaning *absolute value*. The same is true for **ATTN**, **CLR**, **COS**, **SQR**, **SPC**, and many more.

There is one more twist in the use of **LET**. You really do not have to use it. The instructions **LET** A = 10 and A = 10 do the same thing. The **LET** is understood in the second instruction just as the "You" is understood in the sentence "Be careful!" **LET** is not spelled out in many programs in order to save keyboard effort and program memory space.

DATA Statements

If you have a fair number of values to enter, **LET** statements can take up a lot of typing time. There is a way to shorten the entry of numerous values; this is especially useful if they are to be processed one after the other.

A **DATA** statement lists values so they can be read from the list sequentially when they are needed. To see how **DATA** statements can be useful, enter the following program:

```
100 FOR X = 1 TO 10
110 READ Y$
120 PRINT Y$
130 NEXT X
140 DATA YOU,YOUR,I,ME,HE,SHE,HIM,HER,HIS,
    HERS
```

The **READ** statement *picks up* a string value (no quotation marks required for a string value in this one exception) from the list in the **DATA** statement each of the 10 times that the program progresses through the **FOR-NEXT** list.

One other keyword is important in reading **DATA** statements. A **RESTORE** statement resets the **READ** *pointer* to items in **DATA** statements, determining which value is read next. The **RESTORE** command without a line number will cause the next **READ** command executed to pick up the first item in the first **DATA** statement in the program. The **RESTORE** statement with a line number causes the next **READ** statement executed to pick up the item from the first item in the **DATA** statement on or *following* the line of that number.

It's important to understand that **DATA** statements are *not executed*. In the sequence of execution, they are ignored. They mean something only to **READ** statements, and what they mean is determined by **RESTORE** statements.

Math Expressions

BASIC uses standard symbols to indicate addition (+) and subtraction (−). Multiplication is indicated by an asterisk (*) and division by a slash (/). The meaning of the following expression is *A equals five times four plus one*:

```
LET A = 5*4+1
```

The meaning of the following expression is *B equals twenty divided by four minus one*:

```
B = 20/4 - 1
```

The variable A in the first statement, therefore, has a value of 21, and in the second expression B has a value of 4. There is one more math symbol that is frequently seen in programs. Look at the two examples below.

```
LET R = 5*5*5          LET R = 5^3
```

Both these expressions equal 125. The first uses a series of multiplication symbols to obtain the answer. The second expression is read as *five raised to the third power* or *five cubed*. The symbol for powers is an up arrow, which is just above the 6 key on the top row of the keyboard.

When BASIC is instructed to do a series of math operations, it follows a standard sequence. All power (exponentiation) computations are done first. Then positive and negative prefixes are evaluated. Next, multiplication and division are performed. Finally, addition and subtraction are performed. A handy *mnemonic* to remember the standard sequence is *Please My Dear Aunt Sally*, which stands for *Power*, *Multiply*, *Divide*, *Add*, and *Subtract*.

Sometimes the standard order of computing is not the order needed to solve the problem correctly. The order can be changed by the use of parentheses. The computer will do all the computations inside parentheses, regardless of type, before it does the work outside the parentheses. Consider the two expressions below:

PRINT 6*4 – 3 **PRINT** 6*(4 – 3)

The first expression produces 21 because the multiplication is done first, then the three is subtracted from the result. The second expression produces 6 because the subtraction is done first (4 – 3), then the multiplication (6*1). If an expression has several sets of nested parentheses, BASIC does everything inside the innermost set of parentheses first, then works within the next and so on, and finally does the work outside any parentheses. When using parentheses, you should be sure there is one right parenthesis for every left parenthesis; otherwise the computer will report an error when the line is entered.

TYPES OF NUMBERS

The BASIC used in the TI-99/4A can deal with several types of numbers. Here are examples of the types of numbers it accepts:

345.55	positive decimal number
–345.55	negative decimal number
4.33E+5	positive number in scientific notation
4.33E–5	negative number in scientific notation

The last two numbers may require some explaining. Scientific notation is a means of writing very large and very small numbers in a way that conserves space. The number 4.33E+5 is the same as 433,000. You make the conversion by moving the decimal five places to the right. The number

4.33E−5 is the same as .0000433 because you move the decimal point five places to the left. If BASIC encounters a number so large that it cannot display it in normal fashion, it will convert it to scientific notation. In your programming you can also use scientific notation when it is convenient.

A BASIC BREATHER

We'll pause here in our look at BASIC, if only because we have earned a breather. Keyboard symbols, commands, and statements in BASIC are not very difficult to learn, but there is a good bit of detail to get used to through use. We'll discuss BASIC further in Chapter 7, but between now and the time you read Chapter 7 you might want to put some of what you have learned to good use. The best use we can think of is, strangely enough, *play*.

If you want to become familiar with BASIC you are likely to do just that by scanning through this chapter and working with the concepts and the sample programs on your own time. Add to our examples, change them, stretch them, and test them. BASIC is always ready to talk to you inside the TI99/4A. And if you are not certain that BASIC understands what you want it to do, trying telling it what you want. As you've seen, BASIC is most unbashful in telling you when it doesn't understand. But, as you've also seen, BASIC understands a great deal of English language and even more ordinary good sense, your best tools for learning and exploration.

Chapter 7

More BASIC

There are many keywords in TI BASIC we didn't consider in Chapter 6. Most of these phrases are used in more powerful programming techniques. In contrast to the ground rules we have considered thus far, the concepts we consider in this chapter really get us off the ground and start us flying with BASIC.

MAKING DECISIONS AND COMPARISONS

BASIC has several ways of comparing one variable to another and of making decisions. The most important of these are **IF THEN**, **GOSUB**, and **FOR NEXT**.

IF-THEN-ELSE

The keywords **IF**, **THEN**, and **ELSE** are used in combination to enable the computer to make different types of decisions. The following statement is typical of the use of these keywords.

```
50 IF X<N THEN 100 ELSE 200
```

Line 50 looks at the value of X. **IF** X is less than (<) N, **THEN** the computer will execute line 100 next, or **ELSE** it will execute line 200 next. The **IF** clause in line 50 is used to specify a *condition*. If that condition is true, the next line of the program to be executed is the one following **THEN**; otherwise the line with the number following **ELSE**.

The **IF-THEN** statement can involve any combination of the conditions and actions listed, and it can make comparisons that involve values (**IF** S = 5), variables (**IF**X = G), strings (**IF**P$ = "YES"), or expressions (**IF** X/B = Z/A). The following operators indicate the conditions that may be tested in the **IF** clause.

	CONDITION
<	less than
>	greater than
=	equals
< =	less than or equal
> =	greater than or equal
< >	not equal

The **ELSE** clause of the statement is optional. The following statements illustrate the use of **IF-THEN** without **ELSE**.

```
100 IF X<>0 THEN 120
110 PRINT "NOT ";
120 PRINT "TRUE."
```

In the example, if X is non-zero, BASIC prints TRUE, skipping over statement 110 because the **IF** condition is met. Whenever X is zero, however, BASIC prints NOT TRUE; with the condition not met, program execution continues in line-number order, and line 110 is executed to print NOT.

Line 100 of the example has an alternate form that is often confusing to newcomers to programming. The following statement is a direct equivalent of the one shown above:

```
100 IF X THEN 120
```

If your response to the alternate form is *If X in relation to what?*, you are in good form. BASIC, following a convention in older programming language, is being cryptic and implying the relationship. In this case, the relationship is always *not equal to zero*.

Conditions Involving Strings

Although the example above uses numeric values in the conditional clause of the **IF–THEN** statement, comparisons can also involve string values and variables. String comparisons involving the equal sign and the not-equal sign (<>) are relatively easy to understand. Two strings must be exactly the same to meet the equal condition, and a single difference (for instance, CAN NOT versus CANNOT) will cause the not-equal-to condition to be met.

What about comparisons involving the greater-than or less-than symbols? What, for example does the following statement mean?

```
IF A$<B$ THEN GOTO 120
```

Suppose A$ equals THESE and B$ equals THIS. Would the action after **THEN** be executed? Logically, you might assume that since there are five letters in A$ and only four in B$, the condition would not be met, since A$ is not less than B$. However, that is not how the comparison works. When a greater-than or less-than comparison is to be made, BASIC performs the comparison character-by-character in accordance with alphabetic order. The letter A is *less than* the letter B, because A comes first in the alphabet. Thus when the comparison of A$ and B$ is made, the computer will find the first two letters of the string to be equal, but the third letter in A$ is E, while it is I in B$. Because E comes before I in the alphabet, BASIC considers A$ to be less than B$.

FOR-TO-NEXT (and STEP)

We have already used several simple **FOR-NEXT** loops in Chapter 6. We'll use the program listed below to illustrate some of the finer points of **FOR-NEXT** loops. Enter this program so that you can examine it closely.

```
 5 CALL CLEAR
10 PRINT " DOLLARS TO INVEST"
15 PRINT " EACH YEAR"
20 INPUT D
30 PRINT " YEARS TO INVEST"
40 INPUT Y
50 PRINT " RATE OF INTEREST"
60 INPUT R
65 CALL CLEAR
70 LET R = R/100
80 PRINT " YEAR        INVEST        TOTAL"
90 LET B = 1+R
100 PRINT 1;TAB(8);D;TAB(16);D
110 FOR L = 1  TO Y
120 LET B = B*(1+R)
130 LET S = (D*(B-1))/R
140 IF L>=Y THEN 200
150 PRINT L+1;TAB(8);(L+1)*D;TAB(16);INT(S)
200 NEXT L
```

Type in this program exactly as it is listed above, including exactly three spaces between the words in quotes in line 80. Most of the keywords used in the program will be familiar to you.

This program lets you indicate how much money you will invest per year, how many years you will invest that amount, and the interest rate you will receive. BASIC uses this information to compute the accumulated investment per year (under INVEST) and the total value (including accumulated interest) of your investment (under TOTAL). The amount under TOTAL is rounded to whole dollars by the **INT** keyword (you'll learn how **INT** works later in this chapter).

Now, run the program using 600 for the Dollars Invested per Year, 10 for the Years to Invest, and 9 for the Rate of Interest. Here is what you should see on the screen.

```
DOLLARS TO INVEST
?600
YEARS TO INVEST
?10
```

```
RATE OF INTEREST
?9

YEAR          INVEST        TOTAL
1             600           600
2             1200          1254
3             1800          1966
4             2400          2743
5             3000          3590
6             3600          4514
7             4200          5520
8             4800          6617
9             5400          7812
10            6000          9115
```

The screen tells you that, if you invest $600 a year for ten years, you will end up with $9115 of accumulated interest and principle. (This program assumes you do not invest the money until the end of the year; thus no interest is earned during the first year.)

Lines 110 to 200 define a *loop*. It is the heart of the program since it computes the amount of money that will be earned and prints the results. It is called a loop because it is used several times, once for each year you plan to invest.

The **FOR-NEXT** statement controls how many times the computer moves through the loop. The **NEXT** in line 200 defines the lower boundary of the loop. When the computer comes to a **NEXT**, it returns to the line where **FOR** occurs and goes through the loop again. The way **FOR** operates is a little complicated. In this case, L is the *control* variable in the loop (there is no significance to the label L: it could be any other acceptable variable name). The expression on the other side of the equal sign tells the computer where to start and how many times to run through the loop. Line 110 says *Set L equal to 1 (the initial value for the first loop). Each time the computer runs through the loop, increase the value of L by one. When L is equal to Y (the final value), go through the loop one more time and move on to the line immediately following the NEXT statement.* Y is the number of years you plan to invest your money, so there will be one line of results for each year you invest. Since in this program there is no line after line 200 where **NEXT** appears, the computer stops when it finishes the **FOR-NEXT** loop.

As usual there are a few fine points that have not yet been covered. Sometimes you do not want to increase the control variable by one each time the **FOR-NEXT** loop is executed. Here is an example:

```
FOR I = 9 TO 14 STEP .25
```

The **STEP** .25 at the end of the line above tells the computer to increase the value of I only .25 each time the loop is executed. Thus, I would equal 9 the first time, 9.25 the second time, 9.50 the third time, and so on. If this loop was the beginning of a section of the program that calculated and printed the rate of return on investments that drew different rates of interest, the I variable might be used to designate the interest rate. Thus the instruction would give data on interest rates from 9 to 14 in steps of .25. The line **FOR** I = 9 **TO** 15 **STEP** 2 would give the same information, but data would be provided only on interest rates of 9, 11, 13 and 15. The line **FOR** I = 15 **TO** 9 **STEP** −2 accomplishes the same thing. In this case the data for an interest rate of 15 will be printed first, then 13, then 11, then 9. The **STEP** −2 will cause the computer to subtract 2 from the control variable instead of increasing it each time the loop is executed. Another common variation of **FOR–NEXT** loops is to put one inside another. The details of how this works is given in most books on BASIC. If you are using a program that has *nested loops*—that is, loops within other loops the main thing to remember is to be sure the **NEXT** statements are in the proper order. In essence, a loop inside another loop will run through its entire range every time the outside loop runs through one step.

FOR–NEXT loops let you use the instructions inside a loop over and over, the number of times they are used determined by the control values provided. This is a very handy procedure. You will find **FOR–NEXT** loops used in many programs.

There are other techniques for handling repetitive processing, however, one of which uses the keyword **GOSUB**.

GOSUB RETURN

A handy procedure that is common in BASIC programs is the *subroutine*. A subroutine is a set of instructions that are used at several points in a program. The example program below will make the concept of a subroutine clearer:

```
100 CALL CLEAR
110 GOSUB 170
120 PRINT " GUESS MY NUMBER"
130 PRINT
140 INPUT G
150 GOSUB 210
160 GOTO 120
```

```
170 REM RANDOM NUMBER GENERATOR
180 RANDOMIZE
190 LET N = INT(RND * 100)
200 RETURN
210 REM GUESS CHECKING ROUTINE
220 IF G <= N THEN 260
230 PRINT " TOO HIGH: TRY AGAIN."
240 PRINT
250 RETURN
260 IF G = N THEN 300
270 PRINT " TOO LOW: TRY AGAIN."
280 PRINT
290 RETURN
300 PRINT " GREAT, YOU GOT IT!"
310 PRINT " I WILL THINK OF A NEW NUMBER"
320 GOSUB 170
330 RETURN
```

If you run this program, it will print GUESS MY NUMBER at the bottom of the screen. Line 130 is a **PRINT** instruction with nothing to print. It causes BASIC to skip a line each time the line is executed. You then type in a number between 0 and 100 and press ENTER. The computer compares your guess with the number it has generated and tells you if your guess is high, low, or correct. If it is high or low, the computer tells you to try again and asks you to make another guess (don't forget to press ENTER after you do). When you guess the number, the computer tells you GREAT: YOU GOT IT. Then it says I WILL THINK OF A NEW NUMBER and asks you to enter another guess. To stop playing this game you should hold down the CLEAR key.

After clearing the screen in line 100, the program executes the instruction **GOSUB** 170 in line 110. The keyword **GOSUB** tells the computer to stop executing instructions sequentially. Instead it is to go to a subroutine that begins at the line number following the keyword **GOSUB**. In this case the subroutine begins on line 170. Actually line 170 contains a remark that tells us what the subroutine does. If you begin a line with the keyword **REM**, the computer will ignore the line but you can type a message after **REM** that provides information to the programmer.

The subroutine that begins at line 170 generates a random number. The subroutine includes lines 170 to 200. We know line 200 is the end of the subroutine because it contains the keyword **RETURN**.

RETURN tells BASIC that the subroutine has been executed. The computer then returns to the instruction just after the **GOSUB** keyword. In

this case, it is line 120. The computer has branched to a subroutine that did a particular task and then returned to continue normal execution of the program. Line 120 prints GUESS MY NUMBER; then line 140 looks for a number, which you will type in. When you press ENTER, BASIC takes the number you typed in and assigns it to the variable named G (for guess).

Now we come to line 150 where there is another **GOSUB**, this time to line 210. The subroutine includes lines 210 through 330. This subroutine compares the number generated by the computer (variable N) with your current guess (variable G) and gives you either a hint or a congratulatory message. If your guess is too high, the text in line 230 is printed, and the subroutine is terminated with the **RETURN** in line 230. If it is too low (the only other alternative in a not-equal condition), the text in line 270 is printed before the **RETURN**. When your guess is correct (G = N in line 260), lines 300 through 330 are executed to print the congratulatory message, announce that the computer is getting a new number, and actually get the new number. After executing one of these **RETURN** statements, BASIC returns to the instruction just beyond the **GOSUB** that brought it to the subroutine, to line 160.

Line 160 contains a **GOTO** statement that sends BASIC back to line 120. The computer tells you GUESS MY NUMBER and looks for another input. Then BASIC comes once more to the **GOSUB** 210 instruction in line 150, and the comparisons and messages start all over again.

Line 320 does something interesting. We are already in a subroutine, but there is another **GOSUB** in 320. The **GOSUB** there sends the computer to the random number generator subroutine beginning in line 170. When the computer has a new number for variable N, it returns to the instruction just past the **GOSUB** instruction in 320. that is line 330, which is another **RETURN**. This time BASIC goes back to line 160, because line 150 sent it to this subroutine.

There are several advantages to the use of subroutines. You can write all the instructions to do a particular task in one place in the program and then use the **GOSUB** instruction anytime you need that task done. For many people, writing programs is easier if you break the job down into a series of subroutines. (One theory of programming, however, criticizes the use of subroutines. The theory holds that a program should be written so that it executes sequentially, from top to bottom, with no jumping around from one place to another.) In addition, when you must do a particular task at several points in the program, it is very easy to put a **GOSUB** at the appropriate place and avoid writing the same set of instructions at several points in the program.

Subscripted Variables

We now come to one of the last major concepts in BASIC: *subscripted variables*. As noted earlier, variables have names (A, A4, A$, AA, and so on), and they have values. There are two major types of variables—string and numeric; BASIC has several instructions that work only on one or the other.

Up to this point we have dealt with discrete variables with names that are not related, so far as the computer is concerned. Variables such as A1 or A2 or TEST1 and TEST2 are just variables to the computer: the similarity of the names means nothing to the computer. There are no special instructions, for example, that will let you deal with all the variables that have TEST in their name (for example, TEST1, TEST2, TEST3, and so on). However, BASIC does provide a means of dealing with groups of variables. You do it through the use of *subscripted variables*.

Variables with names like A, TEST, and TOTAL3 are just plain, garden-variety variables. Variables with names like A(1), TEST(1), and TOTAL(3) are different matters. The variable A4 is simply a convenient name. A(4), on the other hand, is a subscripted variable; A is the variable name and (4) is the subscript. If you think of the data in a computer as a parade, then A4 might be one lone clown who marches by. A(4), on the other hand, might be one rider in a row of riders. The rider on the far right of the row of riders might be A(1), the one next would be A(2), the next A(3), and the one on the far left would be A(4). If these riders were data in a computer, they would be called a *list* or *one-dimensional array*, since they are an associated group arranged along one dimension.

Now suppose a band comes marching by. Instead of one row, a band is made up of a *matrix* or *array* of people arranged in rows and columns. The person in the first row and first column might be given the variable name B(1,1) while a tuba player in the fifth row and eighth column would be B(5,8). To the computer, the *double subscripts* in the variable names for the band represent the column and row placement of the variables. This is a handy procedure when you have to deal with a large number of variables that are related to one another in some systematic way. In BASIC there are ways of dealing with subscripted variables that cannot be used with regular variables. (It is important, however, to keep in mind that a subscripted variable such as A(2) is an entirely different variable from A2 and A$(2). A$(2) is, of course, a string variable with a single subscript.) Here is an example that illustrates the use of subscripted variables:

```
100 CALL CLEAR
110 PRINT " HOW MANY TOOK THE TEST" ;
```

```
120 INPUT T
130 DIM S(25)
140 FOR X = 1 TO T
150 CALL CLEAR
160 PRINT " TYPE IN SCORE" ;X;" AND THEN PRESS
    ENTER" ;
170 INPUT S(X)
180 NEXT X
190 FOR X = 1 TO T
200 LET TL = TL+S(X)
210 NEXT X
220 LET MEAN = TL/T
230 CALL CLEAR
240 PRINT " THE MEAN SCORE ON THE
    TEST" :" IS" ;MEAN
```

This little program lets you type in the scores students make on a test. It uses the subscripted variable S(#) to store the scores. The first thing the program does is ask you how many scores there are. You enter that as the value for variable T in line 120. If T = 3, that means there must be room in the computer for three scores which, in the program, will be stored as S(1), S(2), and S(3). Line 130 uses a new keyword, **DIM**, to tell the computer that you plan to use a maximum of 25 subscripted variables named S later in the program.

If you use subscripted variables in a program, you will generally want to use the **DIM** instruction to tell the computer what to expect before the subscripted variable is used. **DIM** is the keyword for dimension. It tells BASIC to set aside a section of memory that will be used to store the values of each element of the subscripted variable.

If you do not execute a **DIM** statement before using a subscripted variable, BASIC automatically sets aside space for 11 elements in each dimension (subscripts 0 through 10). If you use fewer elements than that, the only result of not using a **DIM** statement will be that a little memory is set aside for variables that will not be used. If you have more than the 11 elements BASIC sets aside, the result is more serious. You will get the error message BAD SUBSCRIPT IN #. A **DIM** statement will have to be added to the program to make it work correctly. If you inadvertently try to **DIM** the same subscripted variable twice in a program, the computer will balk and give you a NAME CONFLICT IN # error message.

There is a **FOR-NEXT** loop in lines 140 through 180 of the program. The first time through this loop, the computer asks you to type in the first

test score, and it assesses that score as the value of S(1). The second time through the loop, it gets the value for S(2), and so on.

Another **FOR–NEXT** loop in this program occurs in lines 190 through 210. This loop adds up all the scores on the test and puts the total in the variable named TL.

Finally the computer divides TL by the number of scores and prints out the average or mean score on the test. This program illustrates one of the major advantages of subscripted variables: you can use **FOR–NEXT** loops to get the values assigned to the correct variables.

The subscripted variable in the test scoring program is a *one-dimensional* variable. TI BASIC lets you define, or dimension, as many dimensional variables as you like, each with whatever number of subscripts needed. **DIM** R(345), for example, sets aside enough memory for 345 variable names, R(1) through R(345). However, there is one limitation. You must have the memory available to set aside. If you don't, the computer will give you a MEMORY FULL error message. In addition, even if you do have enough memory at the beginning of a program for many dimension instructions, you may not have enough memory to do all that dimensioning, store the program, run the video display, and execute the program. Keep in mind how much memory you have available as you work with **DIM**.

Remember, also, that **DIM** can be used to set up multiple dimensional arrays or matrixes as well as one-dimensional subscripted variables. **DIM** R(2,4) sets aside enough memory for a variable that has two rows and four columns. Up to three dimensions may be specified for an array.

R(1,1) R(1,2) R(1,3) R(1,4)
R(2,1) R(2,2) R(2,3) R(2,4)

Actually, we have not been totally correct in describing how **DIM** works in the example. **DIM** R(3) actually sets aside space for four subscripted variables: R(0), R(1), R(2), and R(3). Many programmers ignore the 0 subscript because some versions of BASIC do not accept it and because most of us are not accustomed to using the digit 0 to indicate something other than nothing. If you are working on a program that takes up most of the memory you hvae available, however, you can save some memory by eliminating the space set up for subscript zero with the following statement:

```
100 OPTION BASE 1
```

If your program encounters a subscript of zero after this statement is executed, BASIC will give you a BAD SUBSCRIPT error message.

DIM and String Variables

Thus far we have been dealing with numeric subscripted variables. Does it work the same way with string variables? Almost. If you give the instruction **DIM** A$(2), the computer will be ready to accept three different strings that can have one or more characters in them. A$(0) might have the value CATS. A$(1) might be AND, A$(2) might be DOGS. Now if you tell the computer to PRINT A$(2), it will print DOGS. What would happen if you told the computer to PRINT A$(0)&A(1)&A(2)? With string variables the computer *concatenates* the strings, that is, it puts them together. The instruction to concatenate the three strings above produces the following word.

CATSANDDOGS

Functions in TI BASIC

Another aspect to be discussed in this chapter is that of functions. A function is a lot like canned laughter on a TV program. Whenever a producer needs laughter, pushing the right button will produce it. Functions work the same way. If you need the square root of a number, for example, you could write a subroutine that uses several BASIC keywords to do the job. However, there is a function in BASIC that will do the job for you. The instruction **LET** M = SQR(A) will take the square root of A and assign that value to variable M. If A equals 81, then M will equal 9. If you type **PRINT** SQR(81) and press RETURN, the computer will print the answer 9 on the display.

There are three families of functions: one that works with numeric variables, one that works with string variables, and a set of general-purpose functions. You have already used several numeric functions in example programs (for example, **INT** and **RND**). Here is a summary of the numeric functions available in TI BASIC:

Numeric Functions

SQR(A) gives the square root of A. A must be a positive number, but can be a variable (**SQR** A), a number (**SQR** 56), or an expression such as **SQR**(X/2*3.5).

ABS(A) gives the absolute value of A. That is, whether A is positive or negative, **ABS**(A) is always positive. If A equals −5, **ABS**(A) equals +5. If A equals +5, **ABS**(A) will again equal +5. A can be a variable [**PRINT ABS**(A)], a constant number [**LET** A = **ABS**(−5)], or an expression [**LET** B = **ABS**(A − 4*6)].

INT(A) finds the integer or whole number part of a number. In TI BASIC, regardless of whether the number is positive or negative, **INT** produces the next lower (positive or negative) whole number. **PRINT INT**(12.3) produces 12 while **PRINT INT**(12.9) also produces 12. However **PRINT INT**(−123.456) and **PRINT INT**(123.00000001) both produce −124.

SGN(A) If A is greater than 0, then **SGN**(A) will equal 1; if A equals 0, then **SGN**(A) will equal 0, and if A is a negative number, then **SGN**(A) will equal −1. The instruction **LET** M = **SGN**(A) will set M equal to 0 if A equals 0, −1 if A is a negative number, and +1 if A is a positive number. **LET** N = **SGN**(B − C+X) will set N equal to 0, +1, or −1 depending on the value of the expression (B − C+X).

RND is used to generate a random number between 0 and 1. For the number to be truly random, a statement that uses the **RND** function must be preceded by a statement containing the **RANDOMIZE** keyword. The numbers generated will be decimal numbers between 0 and 1, such as .0011291504 or .78868103. You can use other keywords to get numbers in any range you need. **LET** M = **INT**(**RND** * 11), for example, will set M equal to numbers between 0 and 10. **LET** M = **INT**(**RND** * 10)+1 will set M equal to whole numbers between 1 and 10.

TRIGONOMETRIC FUNCTIONS BASIC has a set of functions that deal with trigonometric values in radians. **PRINT COS**(34), for example, produces −.8485702748. The four trigonometric functions available in TI BASIC are **COS** (cosine), **SIN** (sine), **ATN** (arctangent), and **TAN** (tangent).

The **EXP** function returns the natural anit-log of the value which follows **EXP**. **PRINT EXP**(4), for example, produces 54.5981501. The **LOG** function returns the natural log of the value which follows **LOG**. **PRINT LOG**(4), for example, produces 1.38629436. **PRINT LOG**(**EXP**(4.2)) produces 4.2.

NOTE: In Chapter 6, the order in which mathematical operations are performed is described. The computer does multiplication before addition unless you have changed the priority order by adding parentheses to math expressions. The math functions described above have the highest priority. Functions are calculated even before multiplication and division.

String Functions

To understand string functions, we must remember that the TI-99/4A stores and processes characters in a code that we touched on briefly in Chapter 1. The computer cannot store the letter A or the keyword **PRINT** directly in memory. It is only capable of storing patterns of electrical

charges that represent numbers. As you might expect, two of the string functions have a great deal to do with our—and the BASIC's—area of strongest possible confusion: the representation of number by character codes and by genuine *computer numbers*.

The string functions are designed to work with strings as they appear in these patterns or codes.

CHR$(#) The **CHR$** function lets you give the computer the value of a number, and it converts the value into the corresponding code for a character. When you enter **PRINT CHR$**(65), for example, an A appears on the screen. When you enter **PRINT CHR$**(90), you should get a Z. The **CHR$** function lets you use the code numbers normally used internally by the computer. Table 7.1 shows the code used by the **CHR$** function. Note that there are codes for all letters, numbers, and symbols normally displayed by your computer.

ASC(X$) This string function returns the code value of the first, or only, character in the string enclosed in parentheses. For example, if A$ is CAT, then **PRINT ASC**(A$) produces the number 67, because 67 is the code for the character C.

LEN(X$) This string function lets you determine how many characters (including spaces) there are in a particular string. If A$ is CAT, then the command **PRINT LEN**(A$) will produce 3 since there are 3 letters in CAT.

STR$ and VAL In one of the example programs in this chapter, the point was made that you cannot perform math operations on strings, nor can you perform string functions on numeric variables. Though this point is true, there are legitimate ways of getting around these limitations using the **STR$** and **VAL** functions. **STR$** can be used to convert a numeric variable into a string variable, and **VAL** will convert a string variable into a numeric variable. Here is an example that shows how these functions work.

```
 5 PRINT " INPUT A"
10 INPUT A
15 PRINT " INPUT A$"
20 INPUT A$
30 PRINT 2*A
40 PRINT 2*A$
```

Now run the program and input the number 48 for variable A and 1234 for string A$. Your screen should first display INPUT A. Respond with 48 and press ENTER. Then the display will reply INPUT A$. To this, you should respond with 1234 and press ENTER. Then the computer will multiply A by 2 and print 96 on the screen. But, when it comes to line 40, it

ASCII Code	Character	ASCII Code	Character
32	(space)	80	P
33	! (exclamation point)	81	Q
34	" (quote)	82	R
35	# (number or pound sign)	83	S
36	$ (dollar)	84	T
37	% (percent)	85	U
38	& (ampersand)	86	V
39	' (apostrophe)	87	W
40	((open parenthesis)	88	X
41	) (close parenthesis)	89	Y
42	* (asterisk)	90	Z
43	+ (plus)	91	[(open bracket)
44	, (comma)	92	\(reverse slant)
45	− (minus)	93	] (close bracket)
46	. (period)	94	^ (exponentiation)
47	/ (slant)	95	_ (line)
48	0	96	` (grave)
49	1	97	a
50	2	98	b
51	3	99	c
52	4	100	d
53	5	101	e
54	6	102	f
55	7	103	g
56	8	104	h
57	9	105	i
58	: (colon)	106	j
59	; (semicolon)	107	k
60	< (less than)	108	l
61	= (equals)	109	m
62	> (greater than)	110	n
63	? (question mark)	111	o
64	@ (at sign)	112	p
65	A	113	q
66	B	114	r
67	C	115	s
68	D	116	t
69	E	117	u
70	F	118	v
71	G	119	w
72	H	120	x
73	I	121	y
74	J	122	z
75	K	123	{ (left brace)
76	L	124	|
77	M	125	} (right brace)
78	N	126	~ (tilde)
79	O	127	DEL (appears on screen as a blank.)

Table 7.1 CHR$ function codes.

will try to multiply the string A$ by 2. It cannot multiply strings, so it prints STRING-NUMBER MISMATCH IN LINE 40.

Now change line 40 as you add the following lines to the program:

```
40 LET B$ = STR$(A)
50 PRINT B$&" STRING NOW"
60 LET B = VAL(A$)
70 PRINT B*2
```

Now run the program again and again give A the value of 48 and A$ the value 1234. As before BASIC prints 96 (2 times 48) from line 30. Line 40 converts the numeric variable A (which equals 48) to a string variable B$. Line 50 prints the string B$ along with STRING NOW. BASIC treats B$ as it does any other string, even though B$ was created from a numeric variable.

In line 60 the string A$ that equals 1234 is converted to the numeric variable B. Line 70 multiplies B by 2 and prints the result, which is 2468. These statements are a pattern for your use of **VAL** and **STR$** to convert string variables to numeric variables and vice versa.

SEG$(A$,#,#) This function assigns a segment of a string variable to another string variable. The overall string from which the substring is copied is A$. The first number in parentheses identifies the character in the string that BASIC begins copying, and the second number tells how many letters are to be copied. **PRINT SEG$("BUSTED",2,3)** produces UST. Many versions of BASIC use the keyword **MID$** for this function.

POS(A$,B$,#) Because it is often important to find specific characters in a string (say the spaces that separate words into sentences), TI includes in its BASIC this *substring search function*. The first string in the expression for the function is the overall string to be searched through. The second string is the one that we're searching for. The number in the expression is the character of the overall string with which we want the search to begin (so that BASIC will bypass previously found substrings, for example). When you issue the first of the following two commands, you'll see that BASIC prints 5 to identify the space that is the fifth character of the string. When you issue the second, it prints 5 again, because it finds the string " " in the first character you specify for the search. When you issue the third, BASIC finds the second space in the string to be the eighth character.

```
PRINT POS(" THIS IS A STRING"," ",1)
PRINT POS(" THIS IS A STRING"," ",5)
PRINT POS(" THIS IS A STRING"," ",6)
```

Other Functions

DEF This function is a truly creative one: it lets you define your own functions and assign your own name to them. It is used in very straightforward fashion, as in the following example in which its use is economical in finding many circumferences.

```
100 DEF CIRCUMFERENCE = 8*ATN(1)*R
110 INPUT " BIG CIRCLE RADIUS? " :R
120 PRINT " BIG CIRCLE CIRCUMFERENCE " ;
    CIRCUMFERENCE
130 INPUT " MIDDLE CIRCLE RADIUS? " :R
140 PRINT " MIDDLE CIRCLE CIRCUMFERENCE " ;
    CIRCUMFERENCE
150 INPUT " SMALL CIRCLE RADIUS? " :R
160 PRINT " SMALL CIRCLE CIRCUMFERENCE " ;
CIRCUMFERENCE
```

In our examples above, we can see that CIRCUMFERENCE becomes a keyword that we can define to be part of TI BASIC. If we don't like the BASIC we have on board already, we can write keywords to our satisfaction.

A Final BASIC Keyword

We will end this chapter with an explanation of a BASIC keyword that has not been used in any of our example programs. However, you may encounter it in programs you use.

ON This keyword is most often used with **GOTO** or **GOSUB**. If it were in a program statement, it would appear as the following:

ON N GOTO 120,130,140,150

BASIC would check on the value of N. If N equals 1, the computer will **GOTO** line 120; if N equals 2, the computer will **GOTO** line 130, and so on. **ON** is a way of selecting alternative destinations for a **GOTO** or **GOSUB** instruction. If N equals 0, is a negative number, or is larger than the number of options available after **GOTO** or **GOSUB**(4 in the example above), the **GOTO** or **GOSUB** will be ignored, and the computer will proceed to the next line of instructions.

NOW THAT WE KNOW

We've finished our general introduction to BASIC. We've covered about all there is to do in the dialect of the language called TI BASIC—all there is except doubtless the most exciting part. Working with numbers and

character strings is all right, but drawing pictures and playing tunes is more exciting. Making video games and programming the TI-99/4A to talk is in the making. See the next chapter for details.

Chapter 8

Graphics and Sound

This is our final chance to look at BASIC, and in this final look, we'll see some of the most exciting features of the TI-99/4A. Without accessories, it comes complete with high-quality graphics and sound-synthesis features. The machine is a natural for drawing pictures and making music, and the instructions you may give it to do these tasks are built into TI BASIC.

We'll also take a brief look at using the TI speech synthesizer. While the unit is not a part of the console, it is a relatively inexpensive accessory that adds another unique dimension to your computer.

THE COLOR SCREEN

The screen of the TI-99/4A is a grid 32 graphics characters wide by 24 characters deep. Unless you change the screen color, it is cyan color during the time BASIC is in command mode (with the cursor awaiting command or program-line input). The screen is a light green, unless you change it, while programs are running.

There are 16 colors you can assign to the overall screen and to graphics characters. The colors are assigned by number, as the following list shows.

Color	Number
Transparent	1
Black	2
Medium green	3
Light green	4
Dark blue	5
Light blue	6
Dark red	7
Cyan	8
Medium red	9
Light red	10
Dark yellow	11
Light yellow	12
Dark green	13
Magenta	14
Gray	15
White	16

Each of the graphics characters you can place on the screen is an eight-by-eight matrix of dots that have any color you choose. In addition, the dots in the matrix that do not make up the character may be assigned another color; in other words, while the screen has a background color for all characters, each of the characters you are using may have its own background color. If the character background color is transparent, the character appears against the screen background.

GRAPHICS CHARACTERS

In the TI-99/4A, no graphics characters are predefined for you. Alphabetic, numeric, and punctuation characters are available, if you want to do graphics with them, but they aren't very satisfactory. Rather than tie you to a rigid set of characters, the designers of this computer let you build your own to fit your needs at any particular time.

The way you build characters is interesting. In essence, you specify each dot in each character you use. A keyword phrase, **CALL CHAR**, lets you specify the dots and assign the code you want to use with the character.

Let's build a character that is an eight-dot by eight-dot checkerboard. To use **CALL CHAR** to do this, we follow the keyword phrase with pairs of digits that identify the dots in the top row, then identify the dots in the second row, and so on down to the eighth. We'll represent the dots that make up our checkerboard character with the number one, and we'll use the number 0 to represent positions in the matrix that form the background of our character. The following figure shows what our character looks like.

Top Row	10101010
	01010101
	10101010
	01010101
	10101010
	01010101
	10101010
Bottom Row	01010101

There is both a boon and a bane to representing these rows of the character to TI BASIC. The boon is that we don't have to type all our ones and zeroes into the **CALL CHAR** statement. The bane is that we have to use a special code, called *hexadecimal numbering*, to represent our ones and zeroes.

Here's the way it works. A hexidecimal number can, in a single numeral, represent a group of four dots, specifying uniquely which ones are on and

which ones are off. Two hexadecimal numerals can represent eight dots. The following table shows how the dot representation by digits is accomplished:

Dots	Digit	Dots	Digit
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

From the table, we see that the digits we need are A and 5. The top row of our checkerboard character is made up of two groups of four dots in a 1010 pattern, so it is made with AA, two hexadecimal A digits. The second row, similarly, is made up of 55. The text of our character definition statement for all eight rows reads as follows:

```
CALL CHAR(128,"AA55AA55AA55AA55")
```

The number 128 is the code we assign to the character. It is a decimal one number higher than the codes that represent alphabetic, numeric, and punctuation characters in the standard ASCII code.

There are two other keyword phrases we need to know about to put our new character on the screen: **CALL HCHAR** or **CALL VCHAR**. Both phrases put the character on the screen at a location you specify. The difference between them is that **CALL HCHAR** lets you specify a number of horizontal repetitions for the character, and **CALL VHCAR** lets you specify a number of vertical repetitions for the character.

Enter the following short program using **CALL CHAR** and **CALL HCHAR**:

```
100 CALL CLEAR
110 CALL CHAR(128,"AA55AA55AA55AA55")
120 CALL HCHAR(12,16,128,1)
130 GOTO 130
```

Run the program, and you'll see our checkerboard character in the center of the screen (in column 12, row 16, as in the **CALL HCHAR** statement). We might explore the **HCHAR** repetition while we're here. Press CLEAR to stop the program, and edit line 120 to replace the number 1 with 5. Now run the program and look at the five checkerboards extending from the

center of the screen to the right. Then replace **HCHAR** with **VCHAR** and watch the characters extend from the center of the screen downward.

Now that you've mastered characters, let's look at colors in greater detail. Individual characters cannot be assigned colors, but groups of characters can. These groups, or sets, are identified by character code, as the following table shows.

Character Code	Set	Character Code	Set
32-39	1	96-103	9
40-47	2	104-111	10
48-55	3	112-119	11
56-63	4	120-127	12
64-71	5	128-135	13
72-79	6	136-143	14
80-87	7	144-151	15
88-95	8	152-159	16

Your character number 128 is a member of set 13. To assign it a color, we use the keyword phrase **CALL COLOR** with the codes for the set, the foreground color, and the background color. Let's insert a line numbered 105 into our sample program to make the colors red on black (as any good checkerboard should be). Remember that dark red is color number 7 and black is color number 2.

```
105 CALL COLOR(13,7,2)
```

And for good measure, let's change the screen background from light green to white (color number 16) with a new line 104, as in the following statement:

```
104 CALL SCREEN(16)
```

Now run the program for a checkerboard of the proper color. Notice that if you still have a repetition number of 5 in the **CALL HCHAR** or **CALL VCHAR** statement, you still have five checkerboards stacked end-to-end.

Animation

The **CALL HCHAR** and **CALL VCHAR** statements may be used to produce animated graphics in much the same way they are produced in motion pictures. If you place a graphics character on the screen, then clear the screen and place the character in a slightly different position, the character will appear to have moved.

Let's animate some small rocket ship characters (though we don't want to write a whole video game just yet). Enter the following program:

```
100 CALL CLEAR
110 CALL SCREEN(2)
120 CALL CHAR(33,"80C07EFFFF7EC0")
130 CALL COLOR(1,11,2)
140 FOR X = 1 TO 32
150 CALL VCHAR(4,X,33,16)
160 CALL CLEAR
170 NEXT X
180 GOTO 140
```

When you run your program, you have 16 yellow rocket ships moving in a wave through the black void of space.

The mechanics of the program are simple. We clear the screen (line 100) and set the screen color to black (line 110). Next we define our rocket ship character, assigning it to character code 33 (replacing the exclamation point of the normal character set, line 120), and we set the color of the ship to be yellow on black (line 130). Then we enter a **FOR–NEXT** loop (lines 140-170) which executes once for each column on the screen. All we do in the loop is display the character, with 16 vertical repetitions, and clear the screen in preparation for the next time it is displayed. Each time the character is displayed, it is positioned one screen column to the right of where it was the previous time. It therefore appears to move.

We can refine our animation by making our positioning increments smaller than the one-character width we have used, but the complexity of the program increases quite a bit. Let's leave it with the basic concept for now.

There is a way to have the TI-99/4A produce smoother graphics with very little increase in program complexity. This method, unfortunately, requires the use of the TI Extended BASIC cartridge. It uses what TI calls *sprites*.

If you have Extended BASIC, you'll have fun working with sprites. Sprites are graphics character you define anywhere on the screen, much as we have defined the rocket ships in the sample program above. The outstanding virtue of sprites is that, after they are set into motion, they continue to move until we stop them. The motion is extremely smooth, and we have no need for **FOR–NEXT** loops to move them. The potential for very high-quality video games with sprites is built into Extended BASIC.

SOUND

Sound and music synthesis is built into TI BASIC in much the same way graphics is. You define notes and noises in your program, and then you use the definitions to produce the sounds through your television set or monitor.

Tones

In an instruction to produce a sound, you must provide three kinds of information: the duration, the frequency, and the volume. The duration is measured in milliseconds ($^1/_{1000}$ths of a second), and it must be within the range of 1 to 4250 milliseconds. The frequency is within the range of 110 Hertz to 44,733 Hertz (from the note A below low C to far beyond the limits of our hearing). The volume ranges from 0 through 30, with 0 producing the loudest sound.

Table 8.1 provides a cross reference between frequency and musical tones:

Let's play middle C for one second at the loudest volume we can send to the television set or monitor. Although we could do this from command

Frequency	*Note*	*Frequency*	*Note*
110	A	440	A (above middle C)
117	A#,B♭	466	A#,B♭
123	B	494	B
131	C (low C)	523	C (high C)
139	C#,D♭	554	C#,D♭
147	D	587	D
156	D#,E♭	622	D#,E♭
165	E	659	E
175	F	698	F
185	F#,G♭	740	F#,G♭
196	G	784	G
208	G#,A♭	831	G#,A♭
220	A (below middle C)	880	A (above high C)
220	A (below middle C)	880	A (above high C)
233	A#,B♭	932	A#,B♭
247	B	988	B
262	C (middle C)	1047	C
277	C#,D♭	1109	C#,D♭
294	D	1175	D
311	D#,E♭	1245	D#,E♭
330	E	1319	E
349	F	1397	F
370	F#,G♭	1480	F#,G♭
392	G	1568	G
415	G#,A♭	1661	G#,A♭
440	A (above middle C)	1760	A

Table 8.1 Musical tone frequencies.

level without a line number, it's better that we do it in a program statement so that we can later experiment without retyping.

Enter and run the following statement:

```
100 CALL SOUND(1000,262,0)
```

We can produce up to three tones at one time. To convert our statement to produce a C-major chord, edit line 100 so that it becomes the following statement:

```
100 CALL SOUND(1000,262,0,330,0,392,0)
```

When you run this statement, you produce the notes C, E, and G simultaneously to make up the C-major chord.

Noise

Of course we don't always want pure tones in our musical notes, so we can add noises for more sound qualities. We can specify one type of noise in addition to our three frequencies in the **CALL SOUND** statement, as we see in the following example:

```
100 CALL SOUND(1000,262,0,330,0,392,0, -3,0)
```

The type of noise we have added is what the TI calls *type 3 periodic noise*, and it adds the quality of vibrato to our C-major chord. There are eight types of noise, each specified with a negative value and a volume number, as in the example above. Types 1 through 4 are periodic noises, types 5 through 8 are white noises. Beyond this skimpy description, we invite you to experiment at your keyboard. One *real-time* noise is truly worth the thousand words that you'd be bored with reading as we attempt a description.

The fascinating relationship between music and mathematics is ready for your exploration inside the TI-99/4A. Type in the following note generator, which plays the first four octaves of notes available to you through sound synthesis without any reference to a table such as the musical-note table above.

```
 10 FOR OCTAVE = 0 TO 3
110 FOR NOTE = 0 TO 11
120 CALL SOUND(200,110*2^(OCTAVE+1)
    *2^(21/12)^(NOTE+1),0)
130 NEXT NOTE
140 NEXT OCTAVE
```

Speech

Speech synthesis is not available with the TI-99/4A as it comes from the package, but it can be added at little expense. A speech synthesizer peripheral is required, and one of three TI software packages is helpful.

Extended BASIC provides speech synthesis for words within its vocabulary (and it spells out words that are not on its list, just as humans do). The following three-line program greets us in Texas style.

```
100 X$ = " HELLO" ::Y$ = " THERE" ::Z$ = " PARTNER"
200 CALL SAY(X$,,Y$,,Z$)
```

(It's true that the vocabulary list that Extended BASIC knows not only fails to include the Texas word *howdy*, but it also pronounces *partner* with a distinct *T* sound. So much for words from Texas.)

A cartridge-program used primarily for converting your computer into a communications terminal is very helpful in producing higher-quality speech than Extended BASIC can produce. Terminal Emulator II lets ordinary TI BASIC access the speech sythesizer to produce words that are not on a list. In addition, you can specify which words in a sentence are to be stressed, thereby giving sentences a more natural sound, and you can learn about *allophones* and other advanced concepts used in the science of linguistics. You might also be interested in the primary use of Terminal Emulator II—speaking words that are transferred to the computer over

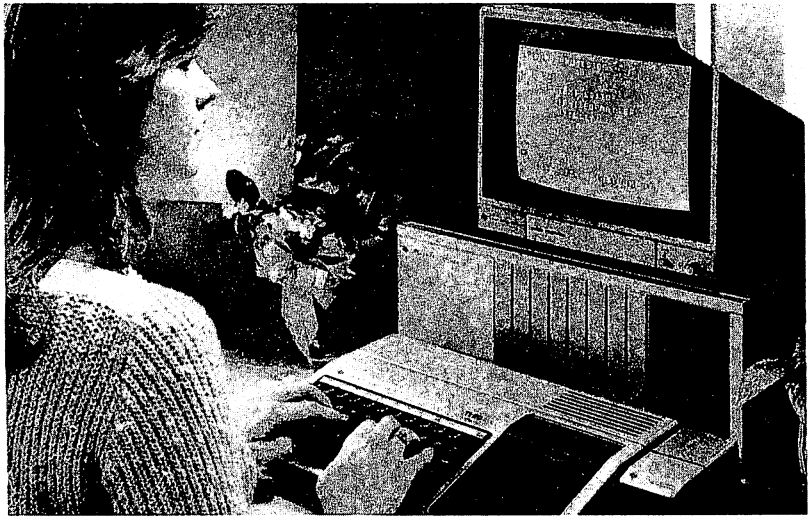

Figure 8.1 TI-99/4A with monitor, expansion box, and speech synthesizer.

telephone lines. It can actually read aloud what you get from TEXNET and your computer friend in the neighborhood.

A dedicated but more limited program to let you produce speech from TI BASIC is Speech Editor, also available in cartridge.

FROM HERE TO THERE, NATURALLY

It's too bad we don't have more space to discuss the making of graphics, sound, and speech, but we have to stop here if we are to bring you an affordable introduction to the TI-99/4A. We hope we've given you at least an insight into what you can create with your machine. By programming the TI-99/4A in BASIC and Extended BASIC you can make your own educational programs and video games, and in the meantime make computer literacy something that is second nature to everyone in your home.

But on the other hand, we can now discuss matters that don't involve so much thought and work. The fact is that *you don't have to program your computer in order for it to give you a great amount of service*. There are literally thousands of programs already written for it by others. In our next chapter, let's see how we can make it easy on ourselves by enjoying the works of others.

Chapter 9

TI-99/4A Software

More than one million TI-99/4A home computers have been purchased, and reasonable projections show that within the next year twice that many will be in use. As far as we know, there aren't that many programmers in the world. We have reason to wonder what people are doing with so many of these machines. We can see the answer when we look at the software available to make the TI-99/4A do many things.

Uses for the computer range from managing the energy used in a home to manage ewe and lamb production, from learning the sounds of the alphabet to playing the stock market, and from fighting galactic wars to playing a Lennon and McCartney song. TI's current advertising campaign stresses the wide availability and variety of software prepackaged for your use: there are more program cartridges available for your TI-99/4A than there are for any other personal computer.

We want to give you an overview of the software here, so that you'll have an idea of what you might be able to do with software you don't have to program. Each piece of software you can plug into your computer's cartridge connector or read into it from cassette or diskette is, to borrow a computerese expression, a *personality module*. It changes what your computer appears to be. With one cartridge, it's a Munch Man video game; with another, it's an electronic cash register. With still another, it's a software development system with the most powerful of languages to help you achieve your programming goals.

We've surveyed the software field and tried many of the packages, but we'll try not to editorialize about the quality of programs. One person's game may be another person's trip into the depths of boredom. A word processing system that does everything for one person may be a frightening experience in comprehension for another. There's no reliable way we can say which programs are for you. What we can do is tell you a bit about some of the programs that seem most significant to us, so you can check them out further if you think they may be what you need.

We'll approach the more serious side of the available software first. Please don't think we're implying that the TI-99/4A is not a fun machine. It is, perhaps *above all*, great for enjoyment and recreation. We mean simply that dessert follows dinner in the order we've chosen. If you're full already, skip a few sections and read about what better suits you. We'll

consider business and home management software, educational, computing, and recreational software.

BUSINESS AND HOME MANAGEMENT SOFTWARE

Although the TI-99/4A is, in amost any form, one of the least expensive personal computers you can purchase, it is also an extremely capable unit, usually at either no extra expense or slight additional expense. It can write letters for you, figure your postage bill for the month, and address envelopes. It can prepare home and business budgets and bill your clients. It can watch your weight, and it can talk to you about foods that make your weight-watching more pleasant. If you want to konw about your home mortgage or your life expectancy, it can tell you something about these topics.

It's in the areas of business and home management that we most often try to assess value received for value paid. In this area, the TI-99/4A is an attractive investment (and, as we'll see later, educational and fun when business dealings and home chores are done).

Secretaries and Clerks

A great deal of software is available to help you write letters (or longer documents such as books), to mail letters to clients and friends, and to file away information for easy recovery later.

Word Processing Packages

The most popular word processing package is the one marketed by TI called TI Writer. It lets you do on the TI-99/4A screen what you might otherwise do much more laboriously on sheets of paper, and it commits the results to paper for you only when you are completely satisfied with them.

When you are typing text into memory, you can insert and delete characters at the cursor. If you wish, TI Writer will automatically indent paragraphs for you. You don't have to worry about the ends of lines, as you do with a typewriter or not-so-well-designed word processors: this one wraps any word that can't be completed by the end of the line onto the next line.

We mentioned in Chapter 1 that the TI-99/4A is limited by screen size to being less than excellent in word processing. TI Writer partially overcomes this limitation by using the 40-by-24 character screen as a *window* that lets you view lines as long as 80 characters. Initially, the window positions itself over the first forty characters of the lines; as you type the line, the window moves to show the center forty characters and finally the

last forty characters. You can therefore type in and view a document as it will later appear on paper, despite the limited screen size.

But typing is only the first of two writing tasks that TI Writer helps you with. The other task is editing. If you need to reverse the order of paragraphs in a letter, have TI Writer pick one up and put it above or below the other. You're forever free of drawing lines to indicate where things should be, of cutting and pasting, and of retyping as penalties you must pay for writing well.

With TI Writer, you can include instructions for your printer to underline, and to overstrike for producing bold characters.

Of course, such a versatile and powerful program requires hardware in addition to the console. You need a printer, an expansion unit, a disk interface card, a memory-expansion card, and a printer.

There are other (generally less expensive) word processing packages that may fulfill your needs, but expect most of them to be slower in responding to your typing and commands than TI Writer. TI Writer is made up of TMS9900 machine language commands, which execute much faster than BASIC, the language used by most other word processors for this computer.

Futura Word Processor stores, reads back from diskette, and edits text files. The package has many interesting features, and it requires two disk drives, TI Extended BASIC, memory expansion, and a printer. This word processor is available from Futura Software (Box 5581, Fort Worth, TX 76108).

TexTiger I and II (TI-99/4A only) are packages that also have many features. These word processors do not require memory expansion or disk hardware, but they require Extended BASIC. TexTiger is sold by Patio Pacific, Inc. (24433 Hawthorne Blvd., Torrance, CA 90505).

TI-Text Writer also lets you create new text in memory, format it, and print or save it on disk or tape. This package requires Extended BASIC, 32K memory, an Epson printer, and either disk or cassette. TI-Text Writer is available from Microcomputers Corporation (34 Maple Avenue Box 8, Armonk, NY 10504).

Typwriter (formerly spelled TI-pwriter) uses any printer and stores as many as 50,000 characters on a diskette or 60-minute tape. This package also requires memory expansion. Typwriter is marketed by Extended Software Company (11987 Cedarcreek Drive, Cincinnati, OH 45240).

Four of the five word processing packages listed above are reviewed in a well-considered article called "Word Processor Market Basket" in the May 1983 issue of *99'er Home Computer Magazine*.

Mailing

Entering information for mailing lists and keeping the information up to date is another job that personal computers can do well. They speed up the job of mailing form letters, meeting announcements, and similar correspondence for either business or personal uses. If you want to spare your secretary some work or make your business run more efficiently, or if you want to help out your church or club, you might look into a mailing program.

Mailing programs are of at least two kinds. The simplest kind keeps a list of names and addresses for periodic printing of labels, envelopes, or both. The more complex kind *merges* information such as names, addresses, and other things we called variables in our discussions about BASIC into form letters so they are personalized.

TI markets three mailing programs. TI Writer, in addition to the tasks mentioned above, does mailing labels or merges information from either the keyboard or a file.

Mailing List is TI's plain vanilla mailing-list program that stores, alphabetizes, sorts, and searches mailing list information, like names, addresses, phone numbers, ZIP codes, and so on. It uses disks and stores about 350 records on each diskette.

TI-Count Mail List is TI's down-to-business bulk-mailing program that maintains 500 to 50,000 names in a list. It can print labels at a rate of 2,000 per hour on a TI 810 printer. It even includes a small word processor for writing and editing letters. The program requires two disk drives, Extended BASIC, a printer, and 32K memory.

Name-It is a companion to TI-Pwriter or Typwriter and produces mailing lists, labels, and cassette or diskette files. It sorts records and, with the companion word processing program, prints form letters with variable entries. The program requires disk or cassette, Extended BASIC, memory expansion, and a printer. Name-It is marketed by Extended Software Company (11987 Cedarcreek Drive, Cincinnati, OH 45240).

Also, mailing-list programs with lesser capabilities are available from several other companies and from users' groups.

Filing

The idea behind a filing (sometimes called a *database*) program is to keep informational records current and organized and sorted or indexed so they can be easily accessed. Business uses include keeping records on customers and clients, sales, inventories, and the like. Home uses include keeping track of automobile maintenance, various kinds of bills, and even birthdays.

There are more than ten filing programs available for the TI-99/4A, and they contain some sophisticated, powerful, and interesting features. Some use cassette-tape storage; others use disks. See Chapter 11 or a list of references in which you can find information about these hard-working programs.

Tables and Charting

We want to mention one more category of programs here, even though it is virtually a "category of one." Multiplan, a *spreadsheet* program written by Microsoft, Inc., a pioneer in personal computing software, is one of the most popular programs marketed by TI. Like VisiCalc, SuperCalc, and other spreadsheet programs, Multiplan permits you to turn your screen into a table, entries in which may contain text or numeric variables. When you change the value of an independent variable somewhere in the table, every entry that has a value dependent on it is also changed. Like TI Writer, Multiplan uses a window on a much larger screen, and your tables can be huge (246 rows by 63 columns). After you have constructed a table, you can print your table or save it to a file (perhaps for use in a document you are typing in by word processing).

Accounting Software

Scores of software packages perform accounting tasks, historically one of the first tasks for which larger business computers were used. These tasks include general ledger, accounts payable, accounts receiveable, inventory, amortization, investment, and other such functions.

Scaled-down versions of these programs are designed to help you manage home financial affairs. They provide you with information-handling convenience in dealing with loans, mortgages, rentals, bills, and the like.

To really describe this category of software and the programs marketed by TI and other companies, we'd have to get into a more specialized discussion than our brief look at software permits. It's important, though to realize that software for these purposes is available.

Agricultural Software

Agricultural software isn't for everybody, but it serves as a good example of specialized but highly supportive software available for TI-99/4A users. If you need to formulate feed or fertilizer mixtures, keep track of breeding and production in a herd of cattle, or analyze your finances in ewe and lamb production, your computer can help with these tasks. Most of these programs are available from Computech Distributing (209 E. Walnut, Springfield, MO 65806).

Home Management Software

Weight control, nutrition, menu or restaurant selection, energy and security management, car-buying decisions, and a few dozen more tasks we perform daily are supported by home management software. Some of the more powerful packages, such as Weight Control and Nutrition marketed by TI, are available from stores, but most of these programs are available from users' groups at very low costs.

EDUCATIONAL SOFTWARE

It's hard to believe after our look at business and home management software, but the TI-99/4A's strongest serious area of application is in education. Among software packages marketed by TI itself, educational programs outnumber any other type of program at least two to one. In general, the software provides individualized learning through the use of the computer keyboard, screen, and sound and speech synthesizers, although some specialized packages help teachers and other authors develop their own software.

The major categories of individualized-learning software correspond with grade levels. The basic skills level covers skills for preschool children (such as learning shapes, colors, numbers, and letters in the alphabet) and elementary through approximately eighth-grade levels of learning in areas such as reading, mathematics, and spelling. High school skills software provides instruction in physics, behavioral science, literature, grammar, and other such areas. There is yet another category that cuts across these three because it contains educational software not at any grade level.

Basic Skills Software

At low level (preschool), the educational software marketed by TI (and others) can be likened to elementary games, except that games are not nearly so educational. Or, it might be liked to the preschool-education segments on *Sesame Street*, such as the "Which of these things belong together?" segments. Increased awareness of colors, shapes, numbers, letters, words, and related concepts (right, left, up, . . .) is the most obvious objective in all of these courses.

But the less obvious object is more active and therefore more effective learning by the child. Even a very young child knows that he or she can't really tell Big Bird the right answer, because Big Bird can't hear and respond. Software for young children takes advantage of the computer's response-recognition capability, and therefore of its ability to respond *individually and effectively* to the child.

If we have any serious criticism to offer of such software for the TI-99/4A, it has to center around its availability. There are very few educational cartridges, tapes, or disks suitable for average preschool children, and the ones that are available aren't sufficiently developed. Even TI's one entry into the field, Early Learning Fun, is not designed to use speech synthesis.

But, beginning with the primary grade levels, the basic skills software (particularly the courseware from TI) becomes much more plentiful and much more thoroughly developed. Early Reading Fun and Reading Fun, both from TI, use speech. A series of first-through-fourth grade mathematics cartridges developed jointly by TI and textbook publisher Scott, Foresman and Company, takes children through addition, subtraction, multiplication, and division with speech support. Even more important for educational effectiveness, a series of four Scholastic Spelling cartridges, developed jointly by TI and school magazine publisher Scholastic Publishing Company, has speech.

Even though speech is important in primary education, other computer aids support the courseware for this level. TI's Math Mission, Alien Addition, Demolition Division, and Meteor Multiplication combine the motivation of playing video games (complete with laser cannons and creeping slime) with instruction in the four basic math operations. A series of six Computer Math games developed by TI and textbook publisher Addison Wesley makes use of music and color graphics in math games for up to three players.

Speech synthesis is used in the TI-99/4A emulation of the TI "Speak and Spell" and "Speak and Math" educational toys. You can have these toys within your computer (at a price higher than the toy price, unfortunately), with the added benefit that the child becomes familiar with the non-toy computer era and accepts the computer as a place where learning is fun.

One educational package deserves special attention. It doesn't teach a particular subject so much as it teaches an understanding of learning through the use of an easily mastered language called Logo.

Logo produces graphics. Statements typed into Logo define the movement of a *turtle* on the screen. The turtle can draw lines and shapes by leaving a trail. And here's where the learning experience really begins. Logo can be instructed to learn the procedure used to draw a line or shape, so that the line or shape can be redrawn when you name it in a statement. Superficially, the drawing of pictures motivates children (and oftentimes adults) to work with this language, but Logo can teach you a great deal about learning (and computer programming concepts as well).

Most of the couseware offered by TI for students at the elementary school level above grade 3 is in the Plato series.

The Plato courseware is one of the most important developments in personal computer education. Plato courseware is a tremendous resource for tried-and-true computer-assisted instruction (CAI) materials. Control Data Corporation, a large-computer manufacturer, has developed this resource over a period of 20 years at a cost of $900 million. Control Data has agreed to provide nine hours of its 12,000 courseware to TI, Atari, and Apple. But, in addition, TI has secured a separate agreement with Control Data to offer an additional 800 hours of Plato courseware to TI-99/4A owners. Overall, TI is converting 117 courses designed to run on the large Control Data computers through telephone lines to terminals into courses that run on your computer from diskettes.

The Plato courses TI has currently announced for the basic skills level include the following:

Basic Number Ideas
Addition
Subtraction
Multiplication
Divison
Fractions
Decimals
Ratio, Proportion, and Percent
Geometry and Measurement
Basic Word Building
Understanding New Words
Understanding What You Read
Thinking About What You Read
Judging What You Read
Parts of Speech
Building and Using Sentences
Spelling and Usage
Capital Letters and Punctuation
Writing Letters (Correspondence)

We've reviewed sample units from approximately 25 percent of the courses, and we're excited about them. Our excitement is not so much from a feeling that every course is excellent, but because we feel the courseware is very good overall and so much of it is *in place* for you to use on your computer.

We should mention that you must have a memory-expansion unit and Plato Interpreter cartridge from TI to use Plato diskettes. It's very likely that you'll be able to borrow the diskettes containing the courseware from public libraries and from public schools if you can't afford to purchase them.

High School Skills Software

Now that TI is offering Plato courseware, the company's previous high school skills courseware is overshadowed. Let's look immediately at the Plato courses available for this level, and then let's look at some important earlier courses that still sparkle in the shadows.

High school courses from Plato include the following:

Basic Number Ideas
Math Sentences with One Variable
Math Sentences with Two Variables
Geometry
Measurement
Special (Math) Topics
Reading
General Reading
Prose Literature
Poetry
Drama
Spelling and Punctuation
Grammar
Diction
Sentence Structure
Logic and Organization
Physics
Chemistry
Earth Science
Biology
Geography
Economics
Behavioral Science
Political Science
History

The most significant of the high school skills programs offered by TI are contained in a series of ten courses developed by the Minnesota Educa-

tional Computing Consortium. Notice, in the following list of these courses, that no overlap occurs with the Plato coursework:

Metric and Counting
Elementary Economics
Elementary Math and Science
Astronomy
Word Beginnings
Exploring
Math Practice
Science Facts
Natural Science
Social Science

These programs are intended for use by junior high or high school students, and they are available only on diskette. To run them, you must also have an Extended BASIC cartridge.

General Education Programs

After our overview of the courseware so far, the general education software may seem skimpy. However, it's pretty substantial if we consider it out from under the shadow of the Plato courseware.

Three courses (Bridge Bidding I, II, and III) from TI are available to strengthen bridge bidding skills for intermediate to advanced players. In the first, you practice bidding your own hand, with the computer bidding your partner's and your opponents' hands. If, in three chances you have at each bid, you don't recommend the best bid, the computer tells you the best bid and explains why it is best. The second course gives you similar practice and instruction in bidding slams, and the third works with you on competitive bidding. These courses are available on diskette.

Two courses are available to tutor you in BASIC. Teach Yourself BASIC requires no peripherals but Teach Yourself Advanced BASIC requires an Extended BASIC cartridge. The programs are available on cassette or diskette.

From TI and other sources, you can also learn to compose music, operate a business in a competitive marketplace, run for president against an opponent, and a great deal more. Many of the general interest courses are simulations or games in which you learn by playing. They are video games with beneficial side effects.

COMPUTING SOFTWARE

Computing software for the TI-99/4A includes four *language packages* and a variety of support programs. Extended BASIC adds considerably to

the power of the TI BASIC included in ROM in the console. TI PILOT provides an easy language for writing *interactive* programs like sophisticated games and course. Pascal provides you with the means to write structured programs that are more organized than BASIC programs. Assembly langugae lets you write programs that execute much faster and much more completely control computer operations than any other language. While a number of the support (utility) programs help in the development of programs in various languages, one of them has two distinctly different functions: it permits the computer to operate as a data terminal, and it provides for the reading and pronunciation of words that are not in the speech synthesizer's vocabulary list.

Extended BASIC

Extended BASIC adds features to the BASIC inside the console. With the enhancements, you can easily enter and display data anywhere on the screen, use sprites for smoothly controlled animation of graphics, write multiple statements on single lines, and have your program take care of errors as they occur (rather than having programs stop running to report the errors to you on the screen). On a more detailed level, the power of certain statements has been increased. The **IF-THEN-ELSE** statement is no longer limited to line numbers when you specify the results of comparisons. You can, for example, assign values as a result of the following equal comparison.

```
100 IF X = 10 THEN LET Y = 0 ELSE LET Y = X
```

Extended BASIC uses about 800 more bytes of console memory than TI BASIC does, so without memory expansion you do notavvae quite so much program and data space. On the other hand, TI BASIC uses memory expansion for program (as well as data) storage. Therefore, if you have memory expansion, you can run much larger programs under Extended BASIC than you can under TI BASIC.

TI PILOT

TI PILOT is a programming language that speeds up and simplifies your task of writing programs that call for much interaction through the keyboard. For instructional programs, you can produce written, graphic, or spoken messages that call for keyboard response. Your program written in the PILOT language then determines the computer's next screen or sound.

UCSD Pascal

The Pascal language package marketed by TI is the entire USCD (University of California at San Diego) P-system. When you run the p-system in

your home computer, it takes complete control of the machine, replacing the BASIC-oriented system in the console.

The p-system requires an expansion box card (or choo-choo train peripheral). This card contains the core of the p-system in read only memory. On the back of the card is a slide switch you can throw to select between the BASIC system and the Pascal system. When you throw the switch, the computer's operating system changes. Menu screens are different: the TI-99/4A takes on an entirely different personality.

Within the p-system, you use the Pascal language to write programs that are *compiled* into something like machine language. This "something like" is called p-code, a fairly fast-running language that is used by the p-code card in the expansion box.

The advantages of Pascal over Extended BASIC are two. First, Pascal provides for more methodical, procedure-oriented programming in which the concepts of program design are very close to the statements in the program. Second, programs written in Pascal run considerably faster than those written in Extended BASIC.

Assembly Language

When speed is the most important consideration in the design of a program, the program can be written in TMS9900 assembly language through the use of TI's editor/assembler package. The assembly language statements you type into the editor in the package are translated by the assembler into TMS9900 machine language, the fastest language that your computer can ever understand.

We have to warn you that assembly language programming is very difficult. You must design your programs to use no more than two-byte variables in each statement. And you also must know the most intricate details about the design and construction of your computer. The editor/assembler manual tells you about most of these details, but it takes a while to master them.

The editor in the package is similar to the editor in TI BASIC, but you can also use your TI Writer to enter assembly language programs more conveniently.

Terminal Emulation and Speech Synthesis

A TI support package called Terminal Emulator II enables the home computer to act as a telecommunications terminal, which can transfer keyboard-entered data or disk files to and from another TI-99/4A. In addition, this package provides for transfer of graphics displays and speech

synthesis of the words that appear on the screen. The computer can read words you type or which it receives from another computer.

The speech-synthesis software included in Terminal Emulator II gives it a valuable secondary use. Read only memory in the speech synthesizer provides a list of words that can be pronounced by Extended BASIC, and Extended BASIC can pronounce no other words. But Terminal Emulator II permits you to use TI BASIC to pronounce any words you produce. It also lets you add intonation patterns (stress, and so on) to make sentences sound more natural.

ENTERTAINMENT

You can obtain several hundred games in cartridge, in cassette, and on diskette for your computer. Sources of these games range from TI itself, to independent third-party software producers, to users' groups. Most games marketed by sources other than TI are programmed in BASIC, and they tend to be slow. An Extended BASIC cartridge is often required for such games, so that sprites may be used to provide more rapid and continuous motion.

We'll look at three of the most popular TI games here: PARSEC, Tombstone City, and Munch Man.

PARSEC

Of the three games, PARSEC is the most novel because it uses voice synthesis. The game is essentially a space travel and battle game, much like Atari's Defender. A female voice, like the voice in the Star Trek television series, provides you output from your spaceship's onboard computer. The voice tells you when enemy invaders are attacking and when asteroids are heading your way. It also gives you compliments about your shooting (when you deserve them, of course).

If you don't have a speech synthesizer on your computer, you can play the game without the voice. Messages appear at the bottom of the screen to provide you with the same information. The objective of the game is to accumulate points by playing through as many levels of difficulty as you can.

At each level of difficulty, you are attacked by waves of six different types of invading ships, each with different methods of annihilating your ship. There is also an asteroid belt you must blast your way through (and if you overheat your laser by blasting away too rapidly, your ship explodes). If your ship is about to run out of fuel, you must carefully fly through a tunnel with ship-shattering rock surfaces to obtain more fuel.

PARSEC is a very involving game, especially with the speech feature activated.

Tombstone City

Like PARSEC, Tombstone City is a game in which you must blast away at the enemy before your ship, a desert schooner, is destroyed. But the game's setting is novel. The game is played in an Old West ghost town during the 21st century.

The city is a grid in the center of the screen where you are safe from alien attack. The city is surrounded by tumbleweeds, cactuses, and morgs (aliens). The cactuses breed morgs, and morgs turn into cactuses when they are shot. Tumbleweeds are obstacles that get in your way and hamper your shooting. If you shoot a morg from inside the city, the cactus he becomes blocks one more entrance into and exit from the city.

The objective of the game is to increase the population of the city. The population increases whenever you shoot a tumbleweed or a morg. But if you shoot all of the tumbleweeds, you get another whole screen full of them to hinder your shooting. If you let a morg get to your schooner, you have one fewer schooner to play with.

The desert can become full of cactuses if you're not careful. If you shoot a morg when it is next to two cactuses, the resulting three cactuses become a single morg. You can clean up the desert this way.

Incidentally, Tombstone City is the example of an assembly language program included on diskette with the editor/assember package. The diskette includes both assembly language and machine language files for the game. So, if you are interested in learning what assembly langugae techniques make a game work, and if you're intersted in playing the game, you can satisfy both interests with the language package.

Munch Man

There is no way for us to avoid saying that Munch Man is like PAC-MAN. A little round critter runs around in a maze, covering every path in it to obtain points and to get a fresh maze to play in.

As Atari's PAC-MAN must flee from four ghosts, Munch Man must flee from four Hoonos, each with a different level of intelligence. From one maze to the next, Hoonos change shape, sometimes resembling traveling pinwheels, sometimes rainclouds, sometimes tornados, and other times resembling no shapes that we have names for.

One more thing: Munch Man really doesn't munch anything in the same way PAC-MAN munches dots. Instead, he lays down a chain to indicate which paths in the maze he has covered.

FROM SOFTWARE TO WHERE...

In this chapter, we've given you an overview of the packaged software products marketed by TI and other sources. There is really much, much more detailed information you might want to have about the software that runs on your TI-99/4A, and we give you a roadmap to sources of more information in the last chapter of this book.

But for now, we need to clarify some mysteries. We've been discussing accessories, or peripherals, that are required when you use many of the software packages. The next chapter focuses directly on these peripherals, clarifying what you can and can't do with each one.

Chapter 10

Selecting Hardware and Accessories

We will begin wiht a few cautions. First, we generally have not included prices when discussing specific products, because computer product prices are notoriously fickle. However, when prices are discussed, they are the prices in effect in late summer of 1983. You may find that by now some prices have increased and others have decreased.

Second, suppliers of computer accessories are a strange assortment. Most companies have good intentions, some provide prompt service and excellent quality, and most deliver products that work. But you should exercise reasonable caution when spending your money, because there *are* a few crooks and a few totally incompetent entrepeneurs out there among the "good guys." Also, because the TI-99/4A is sold in many department and discount stores, you'll often speak to salespeople who know very little about computers and who have confused or garbled half of what they know. Be very cautious about what salespeople tell you.

In preparation for writing this chapter, we tried personally to use as many of these products as possible and to summarize the conclusions of reviews published in magazines. When several products are on the market that do the same thing, we describe the models with which we are most familiar. Products we do not mention may be inferior products, or they may simply have been unavailable for us to review while writing this book. We try to be fair in our reviews and to point out positive as well as negative features in products.

EXPANSION SYSTEMS

As it stands now, your only good choice for a reasonably large complement of peripherals for the TI-99/4A is the expansion box system marketed by TI. For a limited time, the box is available with a 32 kilobyte memory expansion card, a disk controller card, and a disk drive, for a suggested retail price of $550. This price is less than half what it has been, and for good measure TI is throwing in your choice of one of three software packages that usually sell for around $100.

The remaining two cards in the peripheral expansion system are also being discounted during this promotional sale. The RS-232 card is less

than $80, and the p-code card is less than $100. You can have a completely expanded TI-99/4A system, including console and speech synthesizer, at a list price of $850, and if you shop warily, a discount price slightly under $600.

It's likely that when HEX-Bus peripherals are available to replace expansion box cards, TI will discontinue the expansion box system. HEX-Bus units are considerably smaller and less expensive to build than expansion boxes and the cards that fit into them.

HEX-Bus peripherals that have been announced include an RS-232 unit, a wafertape (stringy floppy) unit, a printer/plotter, and a modem (telephone-line interface). In addition, a unit that interfaces the signals from the expansion bus connector on the console to the HEX-Bus units has been announced. No disk unit has been announced, and no HEX-Bus units have been shipped from TI at the time this book was printed.

You can still buy choo-choo train peripherals at some retail outlets and on the used market. These units can be good deals if the price is right. They are compatible with both the expansion box system and the HEX-Bus system; you simply connect the units at the right side of the console. Software for the TI-99/4A "neither knows nor cares" whether a disk controller, a memory expansion unit, a p-code card, or an RS-232 unit is from one expansion scheme or the other.

With the expansion box system, you can add a card or two that are designed, manufactured, and marketed by organizations independent of

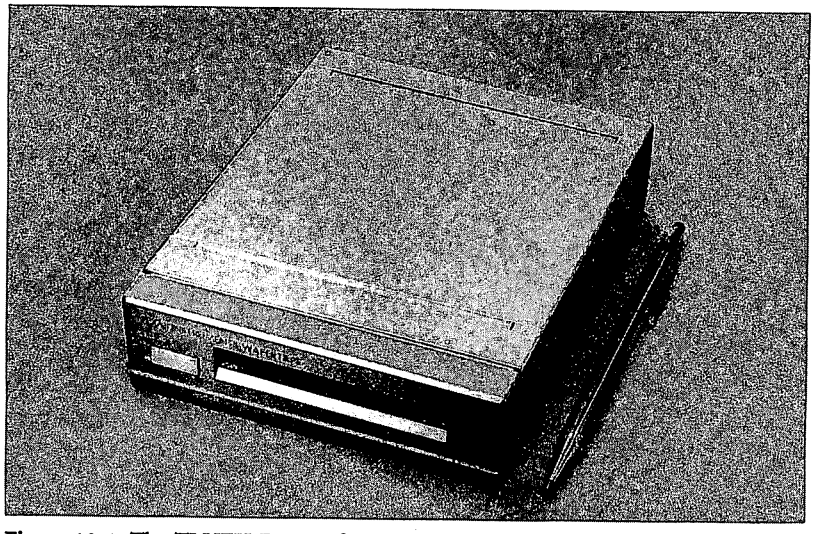

Figure 10.1 The TI HEX-Bus Wafetape Unit.

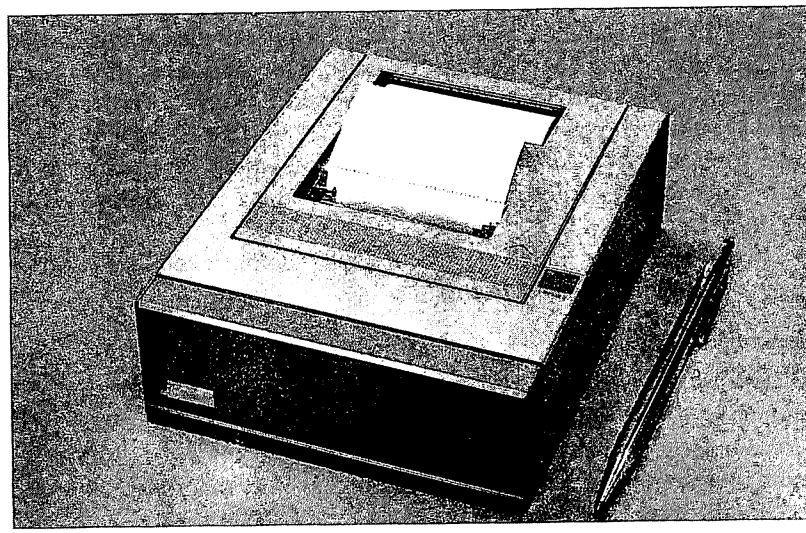

Figure 10.2 The TI HEX-Bus Printer Plotter.

TI. However, with the recent drop in prices, be sure the price of third party accessories is low enough to justify the purchase of non-TI equipment.

If, on the other hand, your requirements for expansion are simple, you may want to avoid the expense of the expansion box altogether. If you need only a printer, the HEX-Bus RS-232 unit and units made by several other

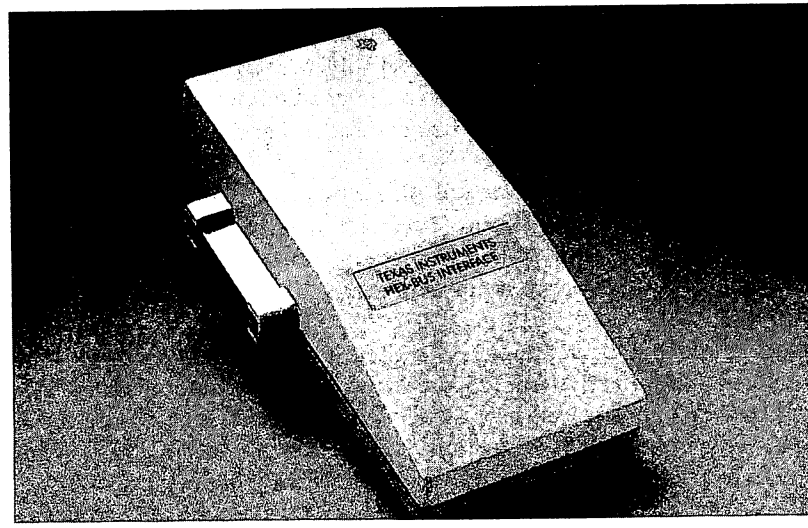

Figure 10.3 The TI HEX-Bus Interface.

manufacturers are available for about the same price as the current RS-232 card, and they do not require the expansion box for connection. If you are a digital electronics buff with sufficient experience, you can try your own hand at building up peripherals like the RS-232 unit described in the June issue of *99'er Home Computer Magazine*. This unit takes under $40 in parts, a few hours of construction time, and a cable to your printer to put you in business.

Let's take a more detailed look at the expansion units and cards according to the functions they perform.

Memory Expansion

The console contains 16 kilobytes of random access memory, and the system can be expanded by another 32 kilobytes. Although the TMS9900 processor is designed to access 64 kilobytes, read only memory (ROM) and input/output devices must use the 16 kilobytes of memory space not used by console or expansion RAM.

TI's current expansion memory is a card that plugs into the expansion box. It contains the full 32 kilobytes.

TI also markets a cartridge it calls MiniMemory. This cartridge contains only 4 kilobytes of RAM, but the RAM is backed up by battery so that it retains its contents when it is unplugged from the console. The RAM in the MiniMemory is primarily useful for developing your own cartridges rather than for significantly increasing the processing power of the system. You can't use the Extended BASIC of editor/assembler cartridge while you are using the MiniMemory cartridge. The MiniMemory cartridge does contain some "code" in ROM that provides TI BASIC with a few of the features of Extended BASIC. The cartridge also contains a line-by-line assembler that permits you to do somewhat limited and somewhat tedious translation of assembly into machine language to be stored in its RAM.

Another 32 kilobyte RAM card that plugs into the expansion box is being marketed by Foundation (74 Claire Way, Tiburon, CA 94920). Foundation also sells a 128 kilobyte memory card but we have not been able to find any software to support more than 48 kilobytes of RAM in the TI-99/4A.

A 32 kilobyte card similar to the one from Foundation is sold by Intellitec Computer Systems (2337 Bonanza Court, Riverton, UT 84065).

Intellitec Computer also has a 32 kilobyte RAM unit that fits like a choo-choo train peripheral into the console expansion connector, saving the expense of an expansion box. Intellitec sells this unit with an RS-232 interface as well.

A separate memory unit that connects in the same way that the Intellitec units do is marketed by Doryt Systems, Inc. (14 Glen Street, Glen Cove, NY 11542).

RS-232 Expansion

Input/output functions may be added to the TI-99/4A by the three types of devices TI markets or by devices from at least two other companies.

Except for physical connections to the CPU, the expansion box card the the choo-choo train units function identically. They provide two RS-232 interfaces. In addition, the expansion box card contains a parallel interface. The RS-232 interfaces may be used for connecting a serial printer, a modem for communication over telephone lines, other computers, and scores of other devices. The parallel interface may be used for connecting a parallel printer that uses a "Centronics-type" interface (see Printers, below).

The HEX-Bus RS-232 unit contains a single RS-232 interface, and it may be purchased with a factory-installed parallel interface as well. Remember, though, that you must also have a separate interface to the HEX-Bus from the computer expansion connector.

A separate RS-232 unit that connects in the same way the choo-choo train peripherals do is marketed by Doryt Systems, Inc. Another such unit

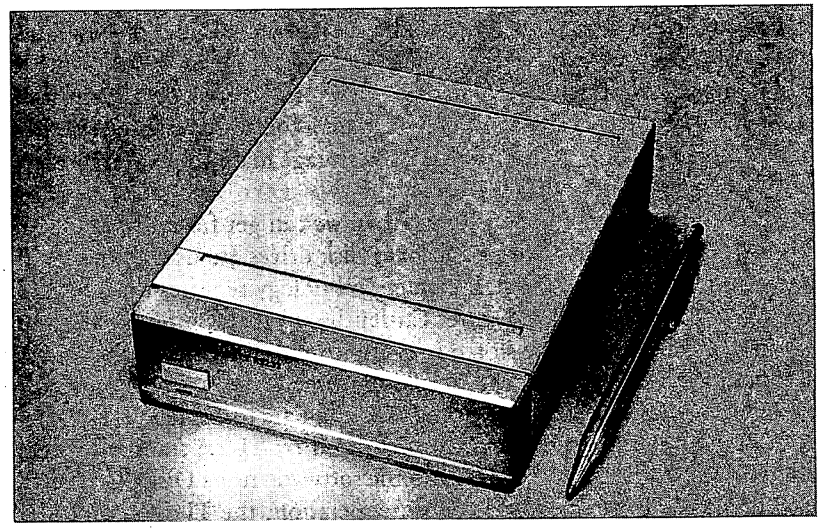

Figure 10.4 The TI HEX-Bus RS-232.

is sold by A. J. International (4023 Sommers Avenue, Drexel Hill, PA 19026).

Disk Controllers and Drives

Let's pause for a moment to discuss a few facts about disks, disk controllers, and disk drives. There are advertisements for a variety of new disk products for the TI-99/4A, and you may want to know what you are getting into if you buy one.

The floppy diskette medium we discussed in Chapter 3 is *written to* or *recorded on* by a disk drive. Except for the geometry involved, the data-writing process is much the same as recording onto magnetic tape. Tracks on the disk are magnetized with encoded data. When the drive *plays back* or *reads* the data, the process is reversed, and the magnetized area are converted back into data.

With standard techniques, approximately 90 kilobytes of data may be stored on one side or surface of a diskette. With more recently developed techniques, 180 kilobytes may be stored by using the *double density* recording technique. Double density data storage can usually be used with the same drives used for standard, single density storage.

With recently developed drives, both sides of a diskette may be used for double sided data storage. Using such drives once again doubles the storage available on a diskette. A double sided drive can store up to appoximately 360 kilobytes on a single diskette in double density format.

The larger the capacity of a drive, the greater the number of programs and data records you can store on it. And for some purposes, 90 kilobytes of program and data just isn't enough. If we want to store a document that takes up 30 single-spaced pages, or if we want a computer-assisted instruction course that lasts 45 minutes, we need storage in double sided format, double density format, or both.

For even larger amounts of storage than we can get from diskettes, we can use *hard disks*. Hard disks and hard disk drives are usually designed into a single unit, because their storage technology requires that they be free of dirt, smoke, and moisture. Currently, up to 10 million bytes (10 *megabytes*) of data may be stored in a single hard disk drive. A hard disk itself is not a direct replacement for a floppy diskette or drive (you must still use a floppy to transport outside programs and data to the hard disk).

Currently, the floppy disk drives marketed by TI are standard single sided drives. Although the drives and the software in the Disk Manager II cartridge are capable of double density operation, the TI disk controller card can only operate single density. For greater capacity with a TI disk controller, you must use double sided drives (also supported by Disk

Manager II), but you must obtain them from a source other than TI. We can't give you all of the technical details for using such drives, but we can caution you to deal with someone you can trust when you obtain double sided drives and instructions for using them. Otherwise, because drives vary in technical details and in quality, your chances for dissatisfaction are pretty high.

One company sells five and ten megabyte hard disk drives and controllers. Myarc, Inc., (Box 35, East Hanover, NJ 07936), markets these models at prices beginning above $2,000. Although we haven't personally used these devices, we do know that they can't operate with the standard TI BASIC operating system directly: it must be *interfaced* to it with the software Myarc provides.

We have one concluding note about disk drives. We don't want to discourage you from using floppy disk drives other than the ones supplied by TI. In the recent past, TI-supplied drives (Shugart model SA400L) have been priced at over twice the price advertised by mail-order suppliers. Our caution applies only to those cases in which you are not sure whom you are dealing with or what you are getting. You may save yourself hundreds of dollars by being cautious.

Printers

The TI impact printer is the only one currently marketed by the company. This printer is, beneath the label, an Epson MX-80FT with interface cabling and instructions particular to the TI-99/4A system. It's a good printer: it has been the standard dot matrix printer for personal computers for the past two years.

The dot matrix printer produces characters made up of discrete dots. The printing looks like that produced by a computer rather than a high-quality typewriter. For letter quality printing, you must look elsewhere than TI.

Your best bet in a low-cost letter quality printer is probably a Smith-Corona TP-1 or TP-2, a printer that TI provides installation suggestions for in its TI Writer documentation. The TP-1 is a daisy wheel printer that produces high-quality print at a relatively slow ten characters per second. Current discount prices for the printer, with serial or parallel interface, are under $600.

Modems: Telephone Interfaces

The word modem is short for modulator-demodulator. A modem modulates audio tones with the data sent out by your computer, effectively converting the data into audio tones with voltage levels that can be transmitted on a telephone line. It also demodulates the tones for receiving

data, converting them into data input for your computer. A modem is a necessary accessory if you plan to use your computer for communicating with other computers and information utilities over the phone.

Modems can be connected to the computer in two ways. Some plug directly into the computer; others plug into a serial port.

The modem currently marketed by TI operates with an RS-232 interface. You plug a cable into your interface to connect it to the home computer. To connect the modem to the telephone line, you press a telephone handset into rubber cups on the top of the modem case. A speaker opposite the telephone mouthpiece and a pickup coil opposite the earpiece transfer tones acoustically and magnetically without any electrical connection between the modem and the telephone line.

The TI modem, actually a version of the better-known Novation CAT, has an under $100 suggested retail price.

The HEX-Bus modem that TI has announced at the same price does not require an RS-232 interface, but it does require a HEX-Bus interface at the console expansion connector. It is a direct-connect modem, which requires electrical connection to the telephone line. One good aspect of direct-connect modems is that they distort the data-transmission tones less than acoustically coupled modems do, and therefore they produce fewer transmission errors. One bad aspect about them is that you should report the direct connection to your telephone company and be billed monthly for it. Many owners, however, do not report owning a modem because the phone

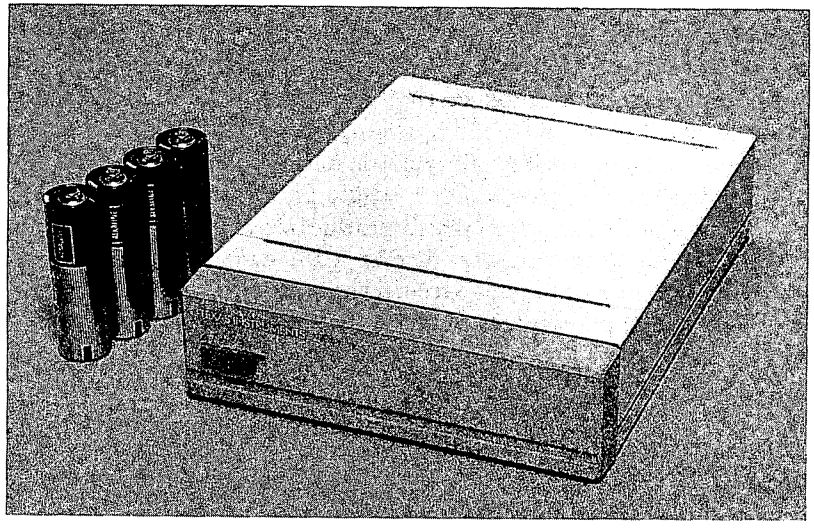

Figure 10.5 The TI HEX-Bus Modem.

company in some sections of the country doubles or triples the basic phone rate under an old rule which was written for businesses that use phone lines for extensive data communications.

Actually, almost any modem on the market will work with the TI-99/4A RS-232 interface. If you are going to be heavily involved in data communication, we suggest that you seriously consider a modem called the Smartmodem, sold by D. C. Hayes Microcomputer Products, Inc. (5835 Peachtree Corners East, Norcross, GA 30092). For about twice the price of the TI modems, you get proven reliability in a direct-connect modem that has been the standard in personal-computer communications for at least two years. You also get automatic dialing (tone or pulse), automatic answering, and "intelligent" computer keyboard or program control of all modem functions. Because the Smartmodem contains its own microcomputer as a controller, you can virtually tell it what to do and receive its response in English words, rather than flipping switches and guess whether what you have done is right or wrong.

MORE INFORMATION ABOUT ACCESSORIES

We've given you a good idea of what accessories for the TI-99/4A are—and will be—like, and yet we feel we've barely scratched the surface of the tremendously detailed information available about products marketed by TI and others. But because we'd like you to be able to gather further information as you wish, we recommend that you read the next chapter. The next chapter provides you with a guide to further sources of information about your computer and its programs and peripherals.

Chapter 11

Sources of Further Information

We hope that this book has served as a good beginning, an introduction, to your use of the TI-99/4A and to the overall field of personal computing. This final chapter will provide you with some suggestions about how to get the additional information you need.

MAGAZINES

99'er HOME COMPUTER MAGAZINE. This monthly magazine is by far the most comprehensive source of current information about your computer. It is devoted exclusively to TI computing products, and most of the articles and reviews in it are about the hardware and software used with the TI-99/4A home computer.

Most of the articles in *99'er Home Computer* are for beginning and intermediate users, and some are for advanced users. It has articles about programming in all of the languages available for the computer, and each issue contains several program listings in these languages.

This magazine has an excellent balance of material. It often publishes interviews with programmers who develop software for the computer; from reading such articles, you can discover how to play a better game of Tombstone City or PARSEC, as well as how to use assembly language speed-up techniques. A typical issue will have articles on programming and reviews of recently released software and hardware. It will also have articles on topics such as word processing, voice recognition and speech synthesis, and the human implications of computers, and articles that include listings of programs you can enter and use.

In addition, the magazine carries scores of ads for products you may want to buy. If you want to know the latest price of a peripheral you've been saving for, this is the place to look.

The current subscription rate for *99'er Home Computer* is $25 per year. You can inquire about it or order it by writing *99'er Home Computer Magazine* (Box 5537, Eugene, OR 97405).

COMPUTE! This is also an excellent magazine for owners of the TI-99/4A. It is published monthly and carries articles on TI computers as well as ATARI, Apple, Commodore, Radio Shack, and Timex Sinclair

computers. The magazine is well edited, very nicely illustrated, and filled with programs you can type in and run on your computer. Most issues contain at least two or three programs that will run on your computer. In addition there are all sorts of articles on how to use the TI-99/4A. Most of the articles are for intermediate or beginning users, but a few are quite advanced.

Compute! also has several regular columns on topics such as educational computing and programming in Logo. This magazine has an excellent balance of editorial material. A typical issue will have articles on programming, reviews of commercial software and hardware, general articles on topics such as word processing, games, and the social implications of computers, and articles that include programs you can type in and use. In addition, the magazine carries hundreds of ads for products you may want to buy.

At $20 a year, *Compute!* is a magazine we strongly recommend. There is a toll-free number for subscriptions (800-334-0868), or you can write **Compute! Magazine** (Box 5406, Greensboro, NC 27403). The people who produce the magazine also publish a line of books.

In addition to the specialized magazines, there are several other computer magazines that regularly publish material on the TI-99/4A. Most of these are available on newsstands where you can browse through them to determine if they fit your needs.

Creative Computing is a fat, fact-filled monthly that covers the general field of personal computing. *Easy Home Computer* magazine also publishes material on the TI-99/4A and is oriented toward novice users. *Computers and Electronics* is another magazine that may be of interest to TI-99/4A owners.

Computers for Everybody, by Jerry Willis and Merl Miller, published by dilithium Press, includes a description of over 20 other magazines that cover different aspects of personal computing.

BOOKS AND BOOK PUBLISHERS

A stroll through the computer section of your local bookstore is likely to show you just how popular your computer really is. There are at least 15 different books about the TI-99/4A in current distribution. Few bookstores are likely to have more than five or so of them, but bookstores are carrying an increasing number of computer books.

In this section we will not try to list all the home computer books currently on the market. There are too many, and the list would be out of date in a matter of months, since new books are appearing regularly. Instead we will note some of the more active publishers of books on the

TI-99/4A. Many of the new books are reviewed each year in the magazines we listed. In addition, several of the suppliers listed in the next section add new books to their catalogs each year:

dilithium Press (P.O. Box 606, Beaverton, OR 97075; phone 800-547-1842). dilithium is the publisher of this book, and it publishes several books on BASIC. The company has over 100 books in print, most of them at the beginning and intermediate level, on personal computing. You can get a free catalog by phoning dilithium at the toll-free number listed above.

Creative Computing Press (39 East Hanover Avenue, Morris Plains, NJ 07950; phone 800-632-8112). This company, now a division of the publishing giant Ziff-Davis, has a good line of books. Several of their publications will be of interest to TI-99/4A home computer owners.

Reston Publishing Company (11480 Sunset Hills Road, Reston, VA 22090; phone 800-336-0338). Reston is a division of Prentice-Hall and is rapidly increasing the number of books it publishes for novice and intermediate computer users.

Howard W. Sams (4300 West 62nd Street, P.O. Box 558, Indianapolis, IN 46206; phone 317-298-5400). Sams is best known as a publisher of rather technical books in the area of electronics, including computers. Recently, however, Sams has begun publishing introductory and intermediate level books on personal computing.

In addition to the books and magazines we have already mentioned, there are two other excellent sources of information on your computer. Several of the *information utilities* mentioned in the Chapter 1 of this book let you search databases for articles, software reviews, and technical notes on the TI-99/4A home computer.

A SPECIAL SOFTWARE DIRECTORY

Texas Instruments publishes a directory of software currently available for the TI-99/4A home computer. The company has attempted to list all software available for the computer by inviting all third-party software authors to describe the products they offer. The directory currently lists approximately 1,000 separate software packages, describes the features of each, and lists the hardware required for running each package. Four indexes get you to the piece of software you're interested in without wasting time.

The *TI-99/4A Software Directory* is available for $5.95 plus shipping charges. To order it, call 800-858-4075 or 800-858-4565.

SUPPLIERS

Many companies produce products for the TI-99/4A. If you are lucky, you live in an area where one or more retail stores carry a wide range of these products. That, alas, is not the fate of most of us. Even when you read a positive review about a great new product or program, it is difficult to buy it locally. A large number of stores simply do not carry the product.

One solution to this problem is to order products by mail. Magazines like *99'er Home Computer* and *Compute!* carry dozens of ads in every issue. Many products must be ordered from the manufacturer, but there are a few distributors who try to stock a wide range of products from many different suppliers. Some of these distributors even publish catalogs that describe the products they carry. Also, mail-order suppliers often sell products at a discount, while local stores do not.

You can save money buying by mail, but be sure that you remember that a local store often provides convenience, local support, assistance, and advice for computer owners. In addition, if a product is faulty, it is usually much easier to correct the problem if you buy locally. We find ourselves buying some products from local suppliers and ordering some by mail. Because companies come and go at an amazing rate in this business, we won't list them here. We can, instead, tell you that your best source of information about mail-order distributers are the advertisements in current issues of the magazines we listed.

SOME FINAL WORDS

This is the final chapter in *How to Use Your TI-99/4A*. We hope you have found the book useful and that you continue to explore the possibilities of personal computing. If you have suggstions you feel would improve the next edition of the book, please write us at dilithium Press. Happy computing!

Index

Here is everything you need to know to successfully and happily operate your TI-99/4A Computer. **HOW TO USE THE TI-99/4A COMPUTER:**

- Introduces you to the computer and its basic components
- Tells you what the components do and how they work together
- Gives you step-by-step instructions on how to set up or install your TI-99/4A and how to get it up and running
- Shows you how to load and save your programs on diskette or standard audio cassettes
- Gives you a brief tutorial on TI BASIC
- Tells you how to type in, use and modify programs published in books and magazines
- Presents you with other sources of information about the computer such as magazines, books and user groups

With an emphasis on practical information, this guide will be your constant companion as you learn to effectively use your TI-99/4A Computer.

0 45078 00395 56135

ISBN 0-88056-135-1

>>$3.95

dilithium Press
GENERAL INTEREST